

TAIPAN

A HISTORICAL ADVENTURE
FOR THE
APPLE® COMPUTER



ART CANFIL, JIM McCLENAHAN, &
KARL ALBRECHT

T A I P A N

A Historical Adventure for the Apple® Computer
(Apple II Edition)

Art Canfil, Karl Albrecht,
and Jim McClenahan

Original Artwork
Chrisann Brennan



Hayden Book Company

A DIVISION OF HAYDEN PUBLISHING COMPANY, INC.
HASBROUCK HEIGHTS, NEW JERSEY

A BASIC Dedication

```
10 DIM WHO4 (4), REASON$ (4)
20 PRINT"TO:"
30 FOR THANKS = 1 TO 4
40 READ WHO$ (THANKS)
50 READ REASON$ (THANKS)
60 PRINT WHO$ (THANKS) ,
70 PRINT REASON$ (THANKS)
80 NEXT THANKS
90 END
100 DATA THE DRAGON, FOR DRAGGING AND DRAGOONING US UNTIL WE
WROTE THIS BOOK, ELDER BROTHER WU, AS A REPRESENTATIVE OF
FINANCIAL REALITIES (IF THE DRAGON HELD THE CARROT - WU HELD
THE STICK)
110 DATA BARBARA FINGER FOR EVERYTHING FROM ENCOURAGEMENT
TO COPYREADING, OUR FAMILIES, FOR THEIR PATIENCE AND
SUPPORT DURING THIS PROJECT
```

Acquisitions Editor: Karen Pastuzyn

Production Editor: Ronnie Groff

Cover design: Paul Perlow

Printing and binding: The Maple-Vail Manufacturing Group

Copyright © 1986 by HAYDEN BOOK COMPANY. All rights reserved. No part of this book may be reprinted, or reproduced, or utilized in any form or by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying and recording, or in any information storage and retrieval system, without permission in writing from the Publisher.

Printed in the United States of America

1	2	3	4	5	6	7	8	9	PRINTING
---	---	---	---	---	---	---	---	---	----------

86	87	88	89	90	91	92	93	94	YEAR
----	----	----	----	----	----	----	----	----	------

Preface — How to Use this Book

This book is written for anyone with a bit of Applesoft BASIC programming knowledge. It is organized so as to allow program lines to be typed into any Apple II computer as you read along.

To make it easier to understand how Taipan works, the program will be shown in a form different from that in which it will actually appear in your Apple II.

Here is an example of a line as it will be presented in this book:

```
280 IF X = 1 AND GP(X1) > C THEN VTAB 18:  
PRINT "YOU CAN'T AFFORD ANY "G$(X1)". ";:  
GOSUB 760:  
GOTO 230
```

In the above example, each logical portion of line 280 is placed by itself.

This is closer to how line 280 will look on your screen after you've typed it into your Apple II:

```
280 IF X = 1 AND GP(X1) > C THEN VTAB 18:  
PRINT "YOU CAN'T AFFORD ANY "G$(X1)  
". ";:GOSUB 760:GOTO230
```

The Apple II computer formats a screen listing of a BASIC program in its own special way. Although you may type in the program as shown above, the computer will add spaces and advance to the next screen line according to its built-in way of doing things. Thus the lines will appear different if, later, you go back to list them.

The most important thing to remember is this: we've used separate lines to show program logic clearly, but you must ignore those separate lines while actually typing this program into your Apple II.

It is a good idea to regularly save copies of the program to disk, especially when you intend to quit working on it for awhile.

Remember that the letter "O" is not the same thing as the numeral "0" — they may look much the same, but to your computer, they're entirely different. Also, some readers may be accustomed to typewriters which use a lowercase letter "l" as a substitute for the numeral "1". Be sure to use the proper key, as the computer knows the difference.

The game Taipan presented in this book is designed to be played on any of the "family" of Apple II computers using Applesoft BASIC, and at least 48K of RAM.

The section of this book entitled "Sea Action" adds graphics routines to make pirate encounters more exciting.



Introduction

In recent years there has been a mushrooming growth in the field of video games. First came simple games in which little white dots bounced back and forth between movable paddles on the screen. Soon these were followed by enormously successful games of shoot-'em-up, eat-'em-up and blow-'em-up.

Entire industries were built upon the growing mass appeal of these games. Investors and corporate conglomerates have poured in millions to keep the video game market constantly expanding. That market has grown, and grown, and grown.

The inevitable has finally happened — people have started to lose interest in video games. Increasingly, people are finding that they "burn out" on games which they find require more of an ability to "twitch" than to think.

This slackening of interest has shaken stockmarkets, retail outlets, and corporate boardrooms. Investors have begun to be cautious about putting their money into electronic games.

But the integrated circuit technology which made video game machines possible also has set the stage for inexpensive home computers. Rather than being limited to a finite number of game cartridges, home computers allow their users to design, write, and execute their own programs, especially in the relatively simple and English-like computer language called BASIC, with which the vast majority of these machines are equipped.

Recently, a very important milestone was reached, almost without fanfare: more home computers are now being sold than video game machines.

Think about this: home computers make possible a type of electronic game which is much less based upon exercising the reflexive speed of a person's brain stem than upon exercising the wonderful gray matter with which humans think.

Among the best of such games are those which simulate a complex environment, and allow the player to interact in a "lifelike" manner with that game environment.

The Authors call such games "Contextual Computer Games".

Contextual Computer Games may be a wave of the future in the field of entertainment and education. Contextual Computer Games take advantage of the computer's speed and power to process information by keeping track of all the elements with which the player must interact. Thus the complexity of the game can be maintained.

But "complexity" can be a misleading word, and a negative one at that. Because a game is complex in its elements doesn't mean it has to be "complex" to play. Baseball is complex in its elements, too. But it doesn't take an Einstein to play it — partly because there is a trained Umpire to say what is supposed to work, and what isn't.

Think of the program of a Contextual Computer Game as being, in one respect, an Umpire. It tells you when you've scored, and it tells you when you're "out". But all you've got to do is play.

The strength of a Contextual Computer Game lies not only in this "automatic Umpire" aspect, but also in its ability to provide unique experiences to even veteran players. A Contextual Computer Game puts together a multitude of factors in constantly changing combinations, to provide challenges much like "real life".

Taipan: A Historical Adventure for the Apple Computer is intended to give the Apple II user three things.

1. An understanding of some fundamental principles of game design
2. A geographical and historical understanding of a particular game context (in this case, Asia and the turbulent China Trade of the 1800's)
3. A step-by-step approach to actually writing a game in BASIC using points 1 and 2 above, including the actual program lines needed to provide a complete Contextual Computer Game

The Authors hope that this book will provide the reader not only with an enjoyable game providing many hours of entertainment, but that the reader will become interested in the game's historical background as well. Hopefully, the reader will be inspired to design original games based upon other historical or fantasy contexts.

May you live in interesting times . . .

Contents

1	Setting the Scene: Hongkong in the Mid-1800's	1
2	Translating the Context into a Game	5
3	Initializing Ourselves into the Past	9
4	Taipan Geography	19
5	A Space-Time Machine	27
6	To Market, To Market . . .	37
7	Cash on the Barrelhead	59
8	Keeping Our Other Options Open	77
9	Let's All Go Down to the Godown	85
10	An International Record Collection	95
11	Brother, Can You Spare a Dime?	111
12	A Gold Watch and a Hearty Handshake	123
13	Anchors Aweigh!	129
14	Piratical Princess of the Eastern Seas	139
15	Stand by the Swivel Guns!	147
16	More Pirates	163
17	Ships of the China Coast	181
18	Current Events	187
19	Action at Sea	199
	APPENDIX A Bibliography	209
	APPENDIX B Playing Taipan	211
	INDEX	213



C. BILZERIAN

T A I P A N

A Historical Adventure for the Apple® Computer

Setting the Scene: Hongkong in the Mid-1800's

CHAPTER ONE

Anyone who has read the remarkable novel, *Tai-pan*, by James Clavell, or *Dynasty*, by Robert S. Elegant, knows something of the exotic and exciting flavor which the words "China Trade" can conjure.

Consider a time and place where fearless and amoral merchant-smugglers of European and American origins struggled and fought, lived to the fullest, and died — and made or lost fortunes in trade throughout the newly opened markets of East Asia.

The doors to trade with China and Japan had been blasted down by the overwhelming military might of the young nations of Europe and America. England, France, Holland, Portugal, and Spain had established important colonial footholds providing local bases for their fleets, missionaries, and merchants. Commodore Perry's intimidating black fleet of American ships had forced the Shogun to open Japan to the traders of the West. Britain's Opium War had inflicted a humiliating defeat upon the Manchu Empire, obliging China to accept trade even in the hated addictive opium. Throughout East Asia, in that violent time, there were warlords and rebels who would pay huge sums for modern Western arms. Europe, in return, hungered for the tea, spices, silks, and gold of

the fabled Orient. Immense fortunes could be made by men without principles to hinder them, if they were brave enough . . .

The China Traders were just such men.

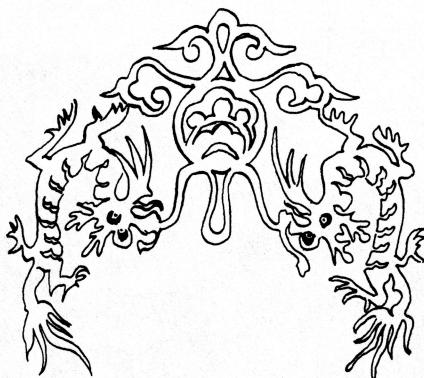
Although the native peoples called them many other, less complimentary names, the Chinese expression *taipan* (or *tai-pan*) was the one seized upon as a title by these Western traders of the China Seas. *Tai* in Chinese means "great" or "gib" or even "supreme". *Pan* means "leader" or "boss" in that language. The China Traders took the name to mean "Supreme Leader". This term still is used for the heads of trading firms from Hongkong to Singapore. But to the Chinese, anyone not Chinese was a barbarian, even if such a person was a "big boss".

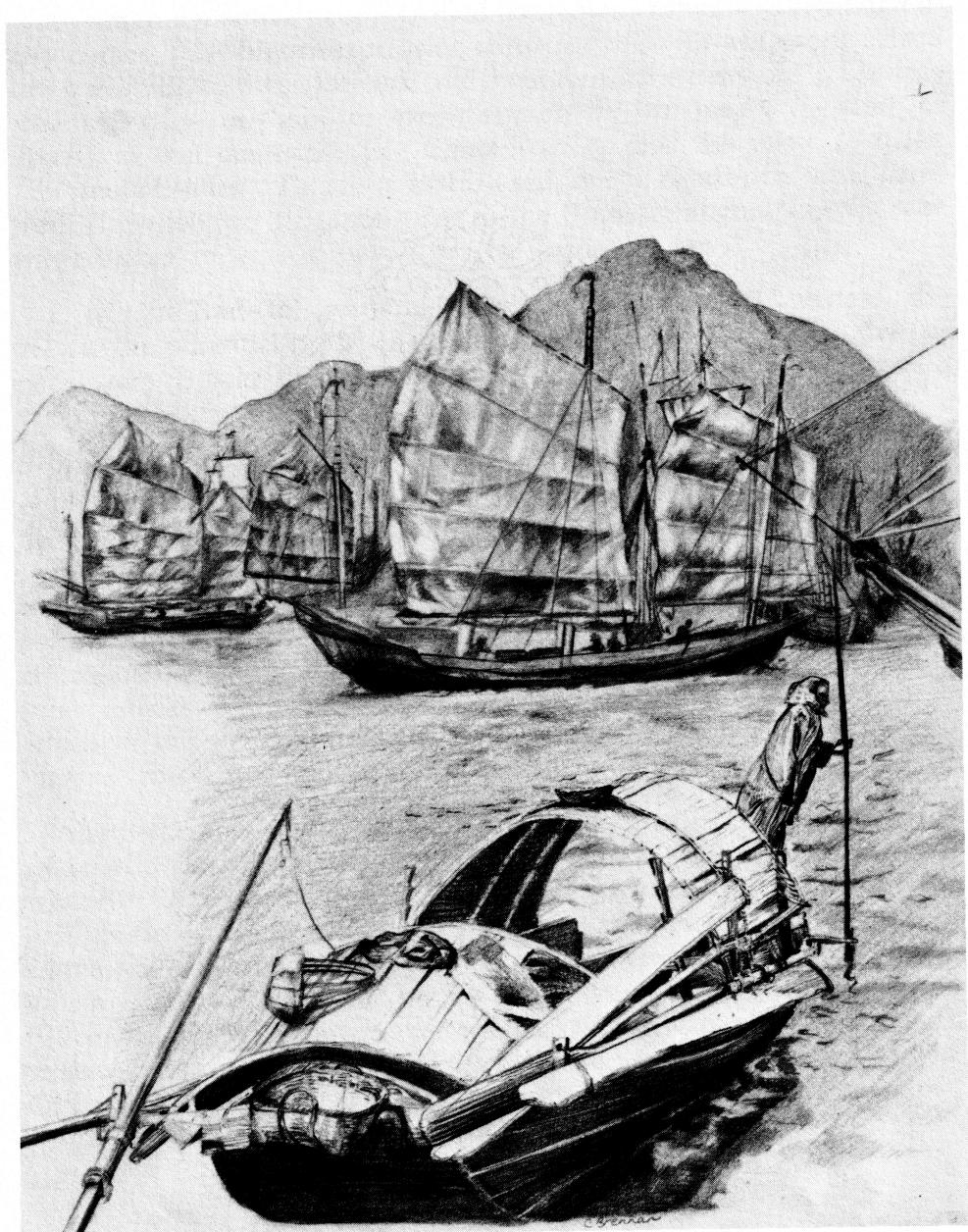
The China Trade followed one of the customary rules of business: the greater the potential profit, the greater the risk. But the risks to the *taipans* were greater than merely going bankrupt: there was always a strong possibility of sudden, and unnatural, death.

Pirates roamed the seas, some of them operating in vast fleets out of ports and coves from Japan to the Malay islands. For the most part, these fleets were under strict military organization, and they were manned by Japanese, Chinese, Malay and European seamen, mercenary adventurers to match the likes of the *taipans* upon whom they preyed. Often the wisest thing a China trader could do was to pay out whatever the pirate chieftains demanded in tribute, rather than face the dangers of deadly pirate attacks. But even if one pirate organization had been bribed not to attack, an independent pirate gang might unpredictably attack. Thus the *taipans* armed their clipper ships, lorcas, and junks heavily.

Another troublesome factor was the presence of the "triads", the underworld secret societies of the Chinese. These triads were to be found wherever large numbers of Chinese lived or worked. Originally founded as Chinese resistance organizations against the hated Manchu rulers of China, the triads had also gradually become organized crime syndicates which wielded violence and money as weapons to exert their power and influence. While Western bankers were often unwilling to extend credit to the *taipans*, sometimes the triads would — at very steep interest rates. Of course, it could be very dangerous not to repay such a loan on time!

This, then, was the world of the China Trade, where incredible risks were taken daily by the taipans, and where enormous riches could be grabbed by the fearless, the resourceful, and the lucky. This is the context around which you are about to build yourself a computer game.





Hong Kong Harbor

T A I P A N

A Historical Adventure for the Apple® Computer

Translating the Context into a Game

CHAPTER TWO

No simulation game can take everything in an environment into account. Just think of the problems involved in trying to create such a "complete" Contextual Computer Game: In the context which we are planning to implement, you'd literally have to take into account the personal psychology, habits, and economic position of every person even partly involved in the China Trade. Not only that, but you'd have to account for the tides and weather conditions in each tiny patch of the China Seas. You'd have to know the location of, and the behavior of, all the wharf rats in Asia. Then what about fluctuations in the financial markets of Europe, the political connivings of the Eunuchs in the Chinese Imperial Court, the effect of frost on silkworms?

These are only a handful of possible items which could be enumerated. Many of these factors may seem unimportant, even irrelevant, at first glance. But they all added to the actual environmental context of the real taipans. So how are we to take them into account?

Obviously we can't put all these things into a single game, but we can do something which better suits our purposes. We can design a game in such a way that, when playing it, we *feel as though* these kinds of factors are part of the program.

How? There are two main approaches to this problem: First, we can include a number of *common* events, such as bad weather and pirate attacks, as fairly regular situations. Secondly, we can create a number of *rare* events, like random robberies and dramatic rises or falls in prices of goods, as *representative* of the vast number of things which could actually happen in the "real world".

Using the built-in "random number generator" of your Apple II computer we can make some events happen quite regularly, while other situations may not occur more than once, if at all, during any particular game. We can set the probability of any event anywhere within a broad spectrum of likelihood.

Trade-offs

We also need to make the context of the game *interactively* "realistic". In other words, the player should have a feeling that the "world" of the game reacts as the real world does. For example, reality constantly presents us with "trade-offs" — situations where we have choices between two or more alternatives, each of which has advantages and disadvantages.

If we are on foot, and need to cross a road against heavy traffic, we might have two choices: One choice might be to jay-walk across the road. The other could be to go down two blocks to a pedestrian overpass, and cross there. With the first choice, we might cross the road much more quickly, thus saving some of our precious time — but we risk not only getting a citation from a police officer, but being killed by a careless driver. With the second choice, we cross the road legally and safely — but we use up more time.

Now let's add another factor: urgency. Suppose that we've just been bitten by a poisonous snake, and the nearest hospital is across the road. Should we jay-walk or take the overpass? Or what if we had all the time in the world that day. Which route then?

In a Contextual Computer Game, we can *vary* this factor of urgency. *Plus* we can vary the danger of the traffic, the pedestrian's ability to dodge cars, and even the structural safety of the overpass! Trade-offs. They're vital factors in Contextual Computer Games, and we'll use them in Taipan.

Player Motivation

Motivating the player is the key to any good game. You don't have to possess a degree in psychology to know some of the things which motivate

people. The desire for power, a lust for money, the drive to gain social status, the pleasure of accomplishing something difficult — all these are common motivations. In *Taipan*, we will attempt to motivate the player with a combination of greed and pride.

Greed is vital, because only with this can the player fit into the role of a *taipan*, the role into which our game will thrust the player.

You may wonder how real greed could be generated in a mere game — after all, there's no real money involved. If you're skeptical about this, then try to remember the last time you played *Monopoly*. After playing for a few minutes, didn't you get just a *little* greedy? If not, you're an exceptional person, and you probably didn't have much fun!

But there's a vital factor here that can't be overlooked by a game designer: in a game (and maybe in life?), wealth, power, or any other reward, will not taste sweet unless there is a struggle to gain it. What would be the purpose of playing *Solitaire* with all the cards in the deck face up? There has to be a conflict, and obstacles to overcome, for any reward to *feel* like a reward.

So in this game we've got to make the player struggle to satisfy greed. That struggle, if successfully carried out, will result in pride of accomplishment.

Difficulty

The degree of difficulty is perhaps the most troublesome factor of all. A game designer has to get it just right. If it's too hard to play, everyone hates the game. Too easy, and people despise it for being trivial. Yet everyone has different standards! It looks as though any game, at best, would appeal only to a certain segment, doesn't it? Not necessarily: by using trade-offs properly, a single game can be a sort of "one-size-fits-all" proposition. It can challenge the bright and the dull, the veteran and the novice player.

How so? There can be so many trade-offs between risky riches and safer plodding, between dangerous shortcuts and secure paths, that there is almost literally something for everybody. We are putting together a game in which anyone who can read and understand words and numbers, and can poke keys on an Apple II computer, has a chance to win — a game in which a PhD with degrees in Asian Studies, Computer Science, and Accounting might lose. A tall order? Maybe. But let's give it a try, together.



T A I P A N

A Historical Adventure for the Apple® Computer

Initializing Ourselves into the Past

CHAPTER THREE

Get your Apple II computer set up! We're about to start typing in our Contextual Computer Game.

Everything must start somewhere — a computer program begins with *initialization*. Here are the first lines of our Initialization routine:

```
5 REM INITIALIZATION (10-50)
```

```
10 HOME:
```

```
A$="
```

```
":
```

```
W$="ELDER BROTHER WU":
```

```
LY$="LI YUEN":
```

```
YS$="YAMATO & SMYTHE":
```

```
TC$="O, S, T, A, P, OR R"
```

```
11 B$="
```

```
"
```

[Note that there are 40 spaces within the quotes which define A\$ in line 10, and 39 spaces in quotes of B\$, in line 11]

```
20 VTAB 4:  
INVERSE:  
PRINT A$:  
HTAB 13:  
PRINT "T A I P A N":  
NORMAL:  
HTAB 13:  
PRINT"_____"
```

[The last "PRINT" statement in line 20 has 17 underline ("_") characters]

```
21 SPEED = 100:  
VTAB 9:  
HTAB 14:  
PRINT "A G A M E I N":  
PRINT TAB( 15);"C O N T E X T":  
PRINT:  
PRINT TAB( 14);"HAYDEN BOOK CO."  
  
22 SPEED = 255:  
VTAB 15:  
INVERSE:  
PRINT A$:  
NORMAL
```

What's going on in lines 10 to 22?

First, we use "HOME" to clear the screen. Next, we create five strings, A\$, W\$, LY\$, YS\$ and TC\$. A\$ consists of 40 spaces strung together. We've created a "string" variable which can be called up and printed at any time: a line of spaces, which we'll use to blot out unwanted stuff from the screen.

WS is used to hold the name of someone we'll get rather too acquainted with in the game, the Triad moneylender, Elder Brother Wu. A player starting the game is immediately in debt to Elder Brother Wu. The player is assumed to have fled Britain rather hastily and under something of a cloud, perhaps signing up as a crewmember on a ship departing Liverpool for the Crown Colony of Hongkong, then jumping ship and starting business under an alias. Of course, no bank in its right collective mind would do business with such obvious riff-raff. So you, as the player, had to turn to the "Elder Brother", or chief, of one of the underground Chinese secret societies. Mr. Wu is willing to make you a loan. Why not? After all, at 100% interest, he can't lose, can he? You've checked around enough to know that Elder Brother Wu's rate of interest is considered quite reasonable by the Chinese standards of the time.

"LY\$" will be used to hold the name of a "mariner" who can be either your friend or your foe, depending upon events and on how you look at them. Let's face it, most folks would consider Li Yuen a pirate. But he would much rather think of himself as the head of a private maritime protective agency, employing a few thousand rough and ready fellows of many nations, the sort of men who might be on the wrong side of the law, were it not for Li. Li has a huge fleet of armed junks and lorchas, and he patrols both coastal waters and the high seas in order to protect his "clients", namely those who are willing to donate to his favorite charity, the building fund of the temple of Tin Hau, a Chinese Sea Goddess. (Odd: the temple never seems to be quite completed . . .)

"YS\$" is also used for names we'll get to know later.

Next, in the interest of saving memory, we set up a string called "TC\$", which is simply part of a prompt we'll use later in the program, whenever the player must choose which item of cargo to deal with.

In line 11, B\$ is set up to be a string of 39 spaces. Like A\$, we'll use this to "clear away" unwanted messages.

Lines 20 through 22 give us our title display.

"VTAB 4" in line 20 tells the computer to position its cursor at V(ertical) TAB 4, in other words, at the fourth line on the screen. (Since we haven't yet given any "HTAB" — or Horizontal TAB — our Apple II assumes we start at HTAB 1, the first column of the line.) The INVERSE statement tells the computer that whatever is displayed should be in inverse mode, in other words, dark letters on an illuminated background. PRINT A\$

gives a line of 40 inverse spaces. This appears as a bar of light across the screen on the fourth line.

Since we've just filled in the entire fourth line of the screen, our Apple now assumes that our VTAB is 5. HTAB 13 thus puts the cursor to the 13th column of screen line five. Program line 20 next has our Apple display the name of the game, TAIPAN, still in inverse lettering, and with inverse spaces between the letters. Our Apple II would continue to forever display words in inverse mode if we let it. We get out of the inverse mode by using the NORMAL statement, then we print a line of underline ("—") characters.

Line 21 of the program continues in much the same way to display the "title page" of our game. But there are some differences. "SPEED = 100" instructs the computer to slow down its rate of display, so that we can actually see the letters light up on the screen one by one. (The lower the number, the slower the display speed. 255 is normal high speed, and 100 is much slower.) We also use the TAB statement in line 21. TAB(14), for instance, works by putting the next PRINT position at the fourteenth column of the screen.

Now we begin to really initialize the program with the next two lines:

```
30 DIM M$(11),G$(5),AP(9,5),
  GG(5),L(9,5),GP(5),V(9):
  FOR I = 0 TO 9:
  READ L$(I):
  NEXT I:
  FOR I = 0 TO 11:
  READ M$(I):
  NEXT I:
  FOR I = 0 TO 5:
  READ G$(I):
  NEXT I

40 FOR I = 0 TO 9:
  FOR J = 0 TO 5:
  READ AP(I,J):
  AP(I,J) = AP(I,J) * 6 ^ (5 - J): NEXT J,I:
  GOSUB 180
```

In these lines we've reserved space for numerical and string variable arrays, and read in string and numerical information from DATA lines which will follow.

But what do these variables stand for, and *why* are we using them? To use an old phrase, what do they have to do with the price of tea in China?

These variables will, in fact, among many other things, be used to keep track of the price of tea in China!

Chinese tea was an item of huge economic importance in Europe, particularly Britain, in the last two centuries. Having been introduced into China from Southeast Asia during the time of the Han Empire (206 BC-AD 220), tea was originally considered a medicine, at first being consumed after being boiled with rice, orange peels, milk, or even onions. Soon, tea was being planted widely in southern and central China. Later, the present method of boiling tea alone in water became the standard practice.

During the 1700's tea became the national beverage of the English. The merchants and officials of the Ch'ing dynasty of China, and their counterparts in Britain, reaped huge profits from the booming trade. Within a century, the tea trade had grown to three-fifths of China's exports.

The British effort to monopolize the trade, and exploit its wide popularity, led to the imposition of tea taxes as high as 100 percent or more. The Boston Tea Party and the slogan, "Taxation without representation is tyranny", resulted. Thus the China tea trade contributed to the causes of the American Revolution.

Line 40 reads in raw values for prices of items of trade, then massages them so that opium, for instance, will cost more than rice.

Another item which we are initializing in line 40 is "arms". As early as 1629, the Chinese Jesuit, Paul Hsu, helped the Ming dynasty (1368-1644) try to hold back the Manchu invaders from the north. To this end, he obtained artillery (and the cannoneers to handle them) from the Portuguese in Macau. Another Jesuit, Father Adam Schall, set up a foundry for the Ming regime in 1636, casting twenty cannon, and naming each after an individual saint. From those times onward, Asians have sought Western arms with much interest.

We're also setting up variables to deal with rice. In China, as in almost all of East Asia, it is the food grain of choice. Like tea, rice is believed to have come to China from Southeast Asia, but in the dim past of prehis-

tory. Rice since ancient times has been the basis of all meals for peasant and aristocrat alike. Steamed rice, fried rice, curried rice, rice cakes, rice pudding, rice wine, rice liquor and even rice paper — this grain has even been used for paying taxes through the centuries in the Orient. In Japan, Inari is the God of Rice, and is symbolized by the fox. During most of Chinese history, the Emperor himself, as the Son of Heaven, would ceremoniously plow a furrow on the Lunar New Year to ensure a good harvest for all his subjects.

But rice never has caught on equally in the West, and since it is a relatively cheap bulk item carried in huge rice barges by the local people themselves, it will stand at the low-priced end among the cargo items with which our game will concern itself.

We will use pepper as a representative of spices in general. (Another item we'll introduce in a moment is silk. Since for playing ease we want to be able to select items by just their first letter, we'll use pepper as a stand-in for all spices.)

Spices are what drove the Portuguese to invade the eastern oceans, competing with the Arabs, who controlled the long-established trade in spice from the Orient to the Occident.

Rounding the south of Africa before the end of the fifteenth century, the Portuguese soon established themselves at Hormuz on the Persian Gulf, defeated the Arab fleets, drove on to take Goa in India, Malacca in Malaya, and eventually to set up "factories" (actually trading posts) on the so-called Spice Islands of the Moluccas in the East Indies.

A desire to have a piece of the spice trade was Spain's primary motivation for financing the voyages of Christopher Columbus, resulting in his accidental discovery of the Americas instead of a short-cut to the East Indies.

Silk is another item we are starting to initialize in line 40.

The ancient Romans, trading through intermediaries, imported so much Chinese silk from the Han Empire, the one-sided trade drained the Roman Empire of much of its gold and silver, doing severe damage to the Roman economy.

Later, during Europe's Middle Ages, Maffio and Niccolo Polo, Venetian merchants, followed the same inland Silk Route which had linked the Chinese and Romans. (Eleven years later in 1271, Marco Polo, Niccolo's

son, joined them in another visit to China. There Marco noted that many European Christians were already present, proselytizing or serving the Kublai Khan as soldiers of fortune.)

The production of silk was an extremely labor-intensive industry. After hatching, 700,000 silkworms weigh in at only a pound. Tenderly cared for and hand-fed mulberry leaves for over a month, that single pound of caterpillars turns into five tones of mature silkworms! By that time, they've eaten twelve tons of mulberry leaves. After all this, only around 150 pounds of silk is produced, and then only after laboriously reeling the silk to make thread. Is it any wonder that silk was expensive?

In the late 1700's the British were attempting to make their growing tea and silk imports less damaging to their economy than silk had been to the Romans fifteen hundred years earlier. They needed export items to balance the trade, so that all their silver and gold would not end up in Chinese hands. They struck upon cotton from Bombay (in the Surat area of northwestern India) and opium (from Calcutta in the Bengal region of northeastern India). Since the British controlled India, they could get these items quite cheaply. The moneys made by selling these items in China balanced the tea and silk trade with China.

As the years passed, opium began to replace cotton in its economic value to the British.

Opium was not unknown to the Chinese, but it had traditionally been used very sparingly and as a medicine, much as it and other opiates are now used in Western pharmacology. But the importation of massive amounts of this narcotic drug by the British, at a time of dynastic decline and decadence in China, caused the opium habit to sweep across that country like a plague.

Opium so altered the balance of trade between the British and the Chinese that a drain on Chinese silver bullion caused a rapid fall in silver's exchange rate with the more common copper currency of China, the "cash". Another effect upon China was the ability of this narcotic trade to corrupt officials. The anti-Manchu Triad secret societies were in the forefront of handling the Chinese side of the trade, often intimidating or assassinating any Chinese official who might try to suppress opium.

With between two and ten million habitual opium smokers in China by the early 1830's, many patriotic Chinese became alarmed, though others, including high officials, connived with the opium traders.

Despite the rampant corruption of the time, one official, Commissioner Lin Tse-hsu (1785-1850), was brave, determined, and incorruptible enough to attack the opium trade head-on. Ordered by the Manchu Emperor to suppress the opium trade, he arrived in the southern Chinese trading port of Canton in 1839. Lin began to close down the city's opium dens, and to arrest and execute Chinese opium traders. He wrote letters to Queen Victoria, with well-reasoned arguments against opium.

In one such letter, Lin wrote, "Suppose there were people from another country who carried opium for sale in England and seduced your people into buying and smoking it; certainly [you] would deeply hate it and be bitterly aroused . . ."

Commissioner Lin impounded 20,000 chests (about 133 pounds of opium each) from the British traders. He then had the opium burned publicly, in an act which is still considered heroic by Chinese of all political persuasions.

But the British were not amused.

England launched the Opium War (1839-1842) in large part to make certain that the trade, so important to the British imperial economy, though physically, mentally, politically and economically destructive to the Chinese, could be maintained at any cost.

It is in this opium trade that the taipans — such as Jardine, Matheson and Company, and Dent and Company — made the lion's share of their fortunes.

Since lines 30 and 40 READ information from DATA lines, let's just skip to those DATA lines right now.

65 REM INITIALIZATION DATA (70-110)

70 DATA HONGKONG,FOOCHOW,
SHANGHAI,NAGASAKI,MANILA,
SINGAPORE,BATAVIA,SAIGON,
CALCUTTA,LIVERPOOL

80 DATA JAN,FEB,MAR,APR,MAY,JUN

81 DATA JUL,AUG,SEP,OCT,NOV,DEC

```

70 DATA OPIUM,SILK,
      TEA,ARMS,PEPPER,RICE

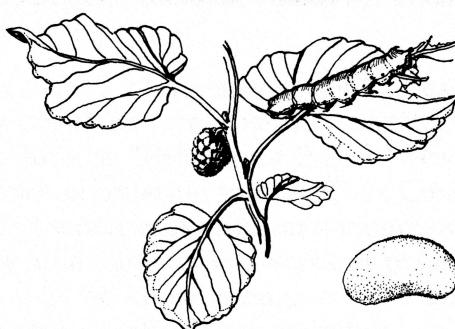
100 DATA 8,2,2,4,5,2,8,2,1,8,4,3,
      9,3,3,6,5,4,6,4,2,8,6,5,
      3,3,4,7,4,4,2,4,5,4,3,5,
      3,4,5,5,2,6,3,1,4,5,3,3,
      1,5,6,3,4,5,3,9,9,1,9,5

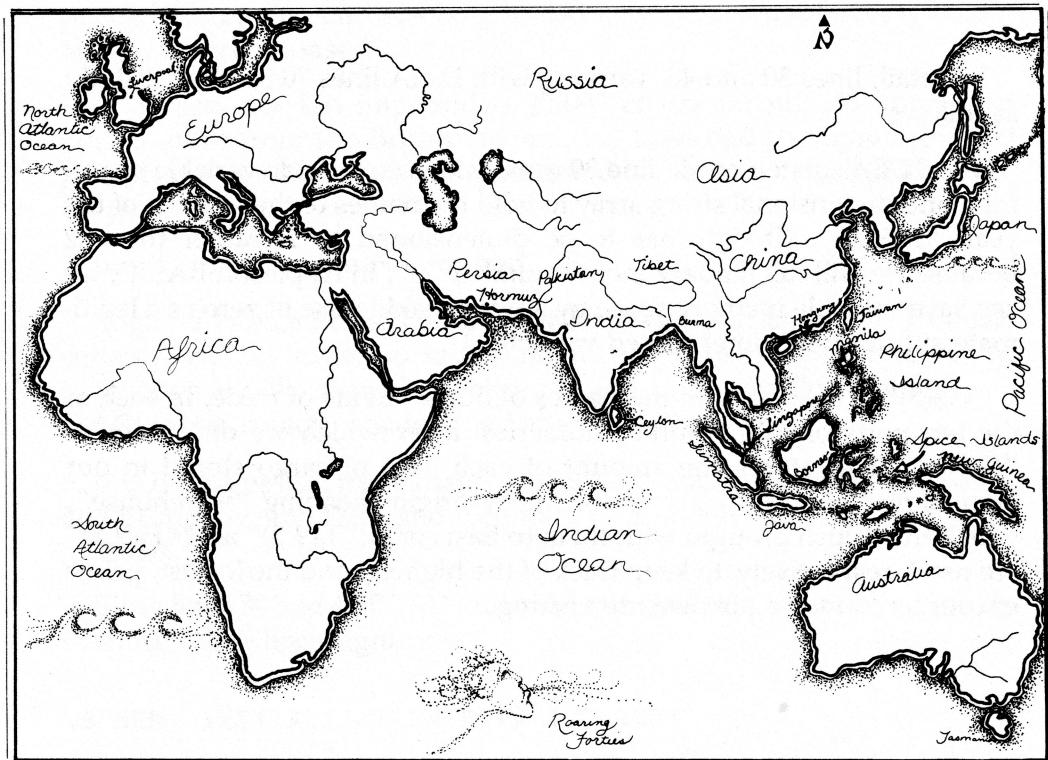
```

In detail, lines 30 and 40, working with DATA lines 70 through 100, do as follows:

The "DIM" statement in line 30 dimensions (sets aside variable space) for a one-dimensional string array to hold the names of the months of the year: "M\$(11)". (It only has to be dimensioned to 11 rather than 12 because we will call January month number "0". In Applesoft BASIC, we can save valuable memory space by always making use of zero as a legitimate subscript of dimensioned variables.)

"G\$(5)" is used to hold the names of our six items of trade, in each of the ten ports of call. Another numerical array which we dimension is "GG(5)". This stores the amount of each item of cargo stored in our Hongkong "godown" (a term of Malay origin meaning "warehouse", which the British brought with them to East Asia). "L(9,5)" and "H(9,5)" are used, respectively, to keep track of the highest, and the lowest, prices encountered in the port we are visiting.





T A I P A N

A Historical Adventure for the Apple® Computer

Taipan Geography

CHAPTER FOUR

We'll be visiting some very exotic ports o' call in Taipan. So the next thing we do in line 30 is start a loop to READ strings from DATA line 70 into "L\$(I)". The loop increments the variable "I" from 0 to 9. This causes the subscript of our "location" string to change upon each READ. The net result is that L\$(0) will hold the string, "HONGKONG", and L\$(9) will be "LIVERPOOL". (The subscripts 1 through 8 will cause the location string to hold the other ports' names.)

Most people we run into don't really know much about geography. If we ask the average person "where is Singapore?", we'll probably get a guess like, "Isn't it in China?"

Let's give you a chance (in case you don't already know all your geography) to be in a position to feel superior to such people. You've no doubt heard of such a thing as "Restaurant French" (useful to the mainly non-French-speaker when ordering food at Chez Continental), or "Take-Out Chinese" (needed when ordering some mouth-watering fried noodles). These are handy little miniaturized versions of languages which many of us learn just so that we know enough to read a menu and order something we love to eat — rather than pointing to a menu item in hopeful

ignorance, and getting some exotic and mysterious thing which, when it arrives, we're not sure is even food.

We're going to cover something the Authors will call "Taipan Geography", a sub-set of world geography which will be limited to ten ports, eight in East Asia, and one each in Central Asia and Western Europe.

Hongkong (usually written in two words as "Hong Kong" these days) is a British Crown Colony. Its official name is Victoria, but hardly anyone has ever called it that.

In July of the year 1839, several drunken British sailors killed Lin Wei-hsi, a Chinese villager. The Superintendent of British Trade, Captain Charles Elliot, refusing to recognize Chinese jurisdiction over British subjects, would not hand over the sailors to Chinese justice. Commissioner Lin Tse-hsu, he of opium-burning fame, stepped up his pressure tactics against the British traders in Canton, forcing them to withdraw first to Macau, then to the coastal islands of South China. The traders settled as squatters on the island of Hongkong in August of 1839.

Hongkong island, when the traders first occupied it, was a mountainous and rocky island with only a few small fishing villages. It lies at the mouth of the Pearl River, downstream some ninety miles from Canton, the traditional trading port of China. Across the wide mouth of the Pearl from Hongkong is Macau, the Portuguese colony which had been established in 1557. Between Hongkong island and the Chinese mainland is a fine natural harbor, usually protected by terrain from the devastating typhoons of the South China Sea. Before Lin could shake a stick at them, the traders, sailors and British officials on the island numbered in the thousands, and about fifty ships were anchored in Hongkong harbor. Commissioner Lin attempted to starve out the squatters, but enterprising Chinese merchants easily got through the blockade.

The Opium war ensued, with the British military taking control of several coastal Chinese cities. The Chinese, soundly beaten by the British, agreed in 1842 to open the so-called "Treaty Ports" of Canton, Amoy, Foo-chow, Ningpo and Shanghai to the British traders. Another concession by the Chinese was Hongkong, which then became officially a British possession.

Eventually, the Crown Colony grew in size, as the mainland peninsula of Kowloon, and the adjacent "New Territories" were added to British control.

Its strategic location, its fine harbor, and the fact that it was outside the

control of the Chinese government, all combined to make Hongkong the headquarters, refuge, and warehouse of the China Traders.

Of the five Treaty Ports, we will only use two, Foochow and Shanghai, as representatives.

Foochow, capital of Fukien province, had also been the capital of the Min kingdom (909-944 AD) during a period when China was broken up into ten kingdoms. After internal rebellions partly disrupted foreign trade to the south along the coast during the middle of the nineteenth century, Foochow became an important import and export trade center. Tea from Foochow was famous throughout the world.

Shanghai is positioned in the middle of China's coast, at the juncture of the East China and the Yellow Seas, and at the mouth of the Yangtse River. The Yangtse gives Shanghai access to a huge slice of China by river barge traffic, through the Yangtse River basin and with connections via the Grand Canal to the Huai and Yellow River basins.

Shanghai became the most Westernized city in China, with traders and banks from many foreign countries setting up business there. As foreign countries wielded greater and greater power in China, it eventually came to pass that signs in Shanghai parks read "No Dogs or Chinese Allowed".

Outside of China itself, the taipans roamed East Asia widely. One place in which they traded was Japan.

For more than two hundred years after 1641, Nagasaki, on the southern island of Kyushu, was the only place in Japan where foreigners were allowed to trade. The Dutch traders (who were the only Westerners allowed to trade with Japan), were kept in near-prison conditions, confined to the tiny man-made island of Deshima in Nagasaki harbor. These restrictions lasted until after Commodore Perry of the United States coerced the Shogun into approving the Treaty of Kanagawa in 1854. That treaty, and a commercial treaty negotiated by the American diplomat, Townsend Harris in 1858, opened Japan to the West, and to the taipans.

Our next port o' call is Manila. Although other Europeans may have visited what today is known as the Philippines, it was Fernando Magellan, a Portuguese in the service of Spain, who attempted to take these islands for the Spanish crown. In 1521, after having rounded the southern tip of South America and crossing the Pacific, Magellan landed on the tiny island of Mactan, just off the central island of Cebu.

Before the Spanish arrived, the Philippines were an island group with separate, small tribal groups and local Muslim sultanates. To the Spanish, who hoped to find spices there, these islands looked to be easy pickings. In a demonstration of macho which might have been admirable (had it not been fatal), Magellan showed the Spanish flag against hostile "savages" led by a local chief, Lapu-Lapu. Magellan grew tired of staying within the protection of his ships' guns, and personally led soldiers onto the beach to find the "enemy". There he found them, just out of the protective cover of his ships' guns. The native warriors had dug trenches on the beach. Bursting forth from the concealment of these trenches, Lapu-Lapu's fighters fell upon the surprised Spanish, killing Magellan. The survivors of the expeditionary fleet returned to Spain under Captain del Cano, completing the first circumnavigation of the globe.

In 1565, Miguel Lopez de Legaspi arrived on Cebu and began a systematic conquest of the Philippines. By 1571, Legaspi had established the Intramuros ("within the walls") fortress at Manila, on the northern island of Luzon. (The Spanish had found a thriving small native town at the site of the Intramuros, named Maynilad, controlled by Raja Sulayman of the newly arrived Muslim faith. In the southernmost islands of the Philippines the Muslim "moros" (moors) remained rebellious, and have never been truly integrated into the Philippines.)

Manila stands on the shore of one of this planet's greatest natural harbors. The Spanish found that the Philippines lacked the spices which they had coveted — they had been shut out of the spice trade by the earlier arrival of the Portuguese in the Moluccas to the south. But the Spaniards soon adapted themselves, using Manila harbor as a transshipment point for a trade in silk from China and other goods from the Orient. These goods were traded for silver coins from Acapulco in Mexico (the so-called "Dollars Mex" which were an international currency in Asia for centuries). This commerce came to be called the Manila Galleon trade. Following favorable winds, the Spanish found the best route to the eastward leg to Mexico was far to the north, where they would pick up the westerly tradewinds. In returning to the Philippines from Acapulco, the Manila Galleons would take a more southerly route. For well over two centuries (1572-1815) the Manila Galleon trade continued almost unchanged. Incredibly, the Spanish never discovered the Hawaiian Islands, though their huge galleons sailed past them to the north and to the south all those years.

Singapore is located just a few miles north of the equator, on an island at the tip of the Malayan peninsula, and at the south-eastern end of the

Straits of Malacca. Since most sea traffic between Europe and East Asia pass through the Straits, Singapore's location is strategic.

The story is told that when an ancient Buddhist prince saw the island, he named it Singa Pura, or "Lion City", because he was supposed to have seen lions there. (This was in the days before eyeglasses: there have never been lions in the area of Singapore.) The Malays who have long lived there called the place Tumasek, or Sea Town.

When the British first came, in January of 1819, to establish a base in Singapore, they found an island little inhabited except by a few pirates. Sir Thomas Stamford Raffles was the brilliant and energetic Imperialist who built modern Singapore. Originally an entrepot port (for warehousing and transshipment of goods) for East and South East Asia, Singapore has grown into a progressive and modern manufacturing center as well.

Perhaps nowhere else in the world can such a population mix be found: Singapore has long had a population of Chinese, Malays, Indians, Pakistanis and Europeans.

The taipans of old would crowd around the Long Bar at the Raffles Hotel, swapping stories and plotting intrigues, perhaps drinking the pink gin concoction known as a Singapore Sling, which originated at the bar. The Long Bar and the Raffles still exist, echoing with faint memories of the time of the taipans.

If you can identify Batavia, you're well ahead of the pack in this Taipan Geography lesson. Because you can't find it on maps these days.

At the west end of the north coast of the heavily populated island of Java, Batavia was established in 1619 by the ruthless Dutchman, Jan Pieterzoon Coen. The city had modern European buildings, yet remained dangerously unsanitary and unhealthy in the tropical climate.

The Dutch competed intensely with other Europeans, stealing the spice trade from the Portuguese and setting up a system of "factory" posts linking the Netherlands with their eastern enterprises. Dutch posts were established in India, Ceylon, Burma, Siam, Cambodia, Vietnam, Taiwan and Japan.

The Dutch East India Company (Vereenigde Oostindische Compagnie) soon dominated the Indonesian islands, developing a form of colonialism which may have been the harshest ever devised. In an effort to control spice trade completely, the Dutch destroyed all clove trees throughout Indonesia, growing them only on the island of Amboina, and generally

would allow only certain spices or crops, such as nutmeg, to grow on selected islands. The effect of this imposed monoculture was often to impoverish or starve the local populations. But this was of little concern to the profit-seeking Dutch Company. Chinese middle-men were used by the Dutch to farm taxes and crops from the native people.

When the Japanese invaded the Dutch East Indies during World War II, they renamed the city "Jakarta", which it had been called in ancient times. At the end of the war, the British forces attempted to re-establish the Dutch in Indonesia. When the British landed in the fall of 1945, they at first found the Indonesian nationalists fighting the Japanese. The British, rather than disarm the Japanese troops, instead ordered them to continue fighting the Indonesians, in order to put the Dutch colonial administrators back in the saddle. The attempt of the Dutch to regain control of Indonesia was long and bitter, but resulted in an independent Indonesia in 1949, with its capital at Jakarta.

Though Batavia is better known to the world as Jakarta, our next port, Saigon, is better known by that old name than by the official new one, "Ho Chi Minh City".

French interests in Vietnam began as early as 1648, through well-organized Jesuit missionary work. Known in the 1800's as Annam (from the Chinese words *An Nan*, meaning "Pacify the South"), Vietnam had gradually become a French colony between 1858 and 1885. Saigon, situated not far from the sea on the navigable Saigon river, was rarely a capital during the history of Vietnam. But it was certainly the most vigorous commercial center of that country. The administrative center of Cochin China, as the French called the southern rice-growing delta area of Vietnam, Saigon (and its Chinese quarter, Cho Lon) became one of the busiest ports in Southeast Asia.

Calcutta, in the Bengal region of India, was among the British Empire's major ports, beginning with a factory post there in 1690. At first, the trade from Calcutta was mainly in textiles, sugar and raw Bengal silk, traded to Europe. This commerce developed into the so-called "country trade", with private British traders carrying the goods. By the late 1700's, this "country trade" was turning toward China as a market, and to opium as a profitable trade item. Opium was harvested in the Bengal as a monopoly of The Honorable British East India Company (originally named "The Governor and Merchants of London Trading into the East Indies", and granted a monopoly on trade in the Pacific and Indian

Oceans by Queen Elizabeth in 1600). The Company auctioned the opium to private traders in Calcutta.

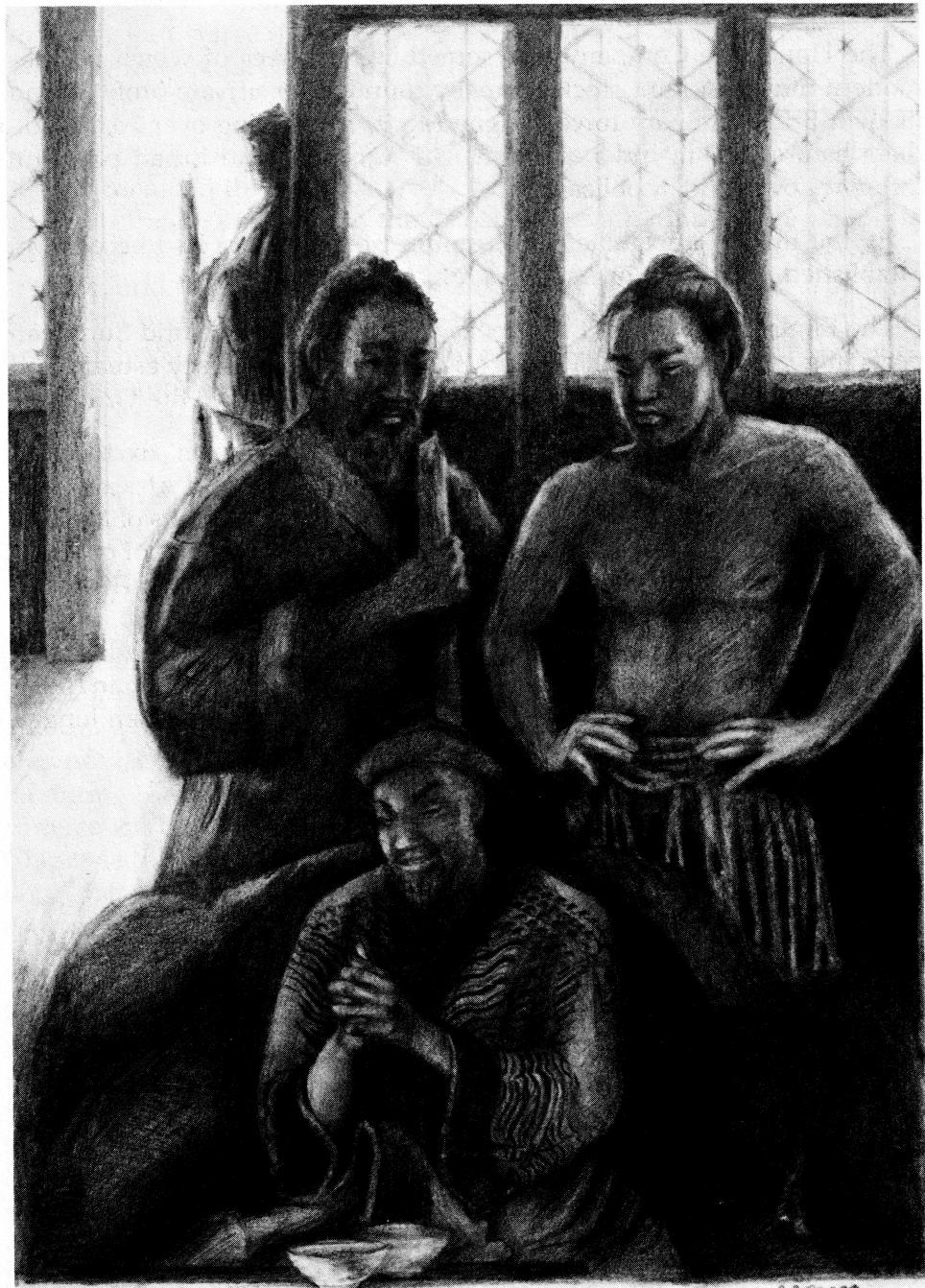
The Honorable Company was something the likes of which boggles modern minds. A joint-stock company founded for private profit, it had its own British military force assigned to it, numbering over 20,000 soldiers before 1800 in India alone. It had warships, and it had powerful influence over British policies.

By the time of our game, 1860, opium from Calcutta had become an established institution in Asian trade.

The English port city of Liverpool will represent British and European ports in general in *Taipan*. Through this port on the Mersey estuary and the Irish Sea, passed a large share of Britain's trade with the Orient.

A sailing ship in the 1860's would take roughly from six to eight months to reach Hongkong from Liverpool, sailing around Africa's Cape of Good Hope, into the Indian Ocean, and through the Straits of Malacca into the China Seas. The Dutch had discovered the technique of continuing directly eastward after rounding the Cape of Good Hope. This practice allowed the "Roaring Forties" (high winds which whip around Antarctica in an eastward direction at about forty degrees south latitude) to push ships quickly toward East Asia, without the need to stop in India. Before using these winds, the eastward passage had taken even longer.





Elder Brother Wu at the office

T A I P A N

A Historical Adventure for the Apple® Computer

A Space-Time Machine

CHAPTER FIVE

We're not yet finished laying the foundation for the environment of our game. The next loop in line READs in the abbreviations of the months, from "JAN" to "DEC" into "M\$(I)", from DATA line 80. Likewise, the last loop in this line READs in DATA from line 90, naming the items of trade.

Line 40 sets up "nested" loops (one within another) which READ in DATA from line 100, then run it through a funny-looking formula to arrive at a raw base price for each item in each port.

That funny-looking formula takes a simple, one digit number from line 100 and multiplies it according to the type of item. For instance, opium will always cost more than rice. We're using this formula, rather than just putting the whole numbers in DATA statements, simply to save memory space. (Saving space in microcomputer memory is an art and science which is as necessary as it is evil.)

These base prices will later be further processed upon first arriving in each port, to give us the actual prices. This is done after the final statement of line 40, "GOSUB 180".

Let's skip to line 180 so we can see what it does . . .

```

175 REM PRICE VARIATION SUBROUTINE (180)
180 FOR I = 0 TO 5:
    GP(I) = INT (AP(L,I) + ( RND (1) * AP(L,I))): 
    NEXT I:
    RETURN

```

Line 180 takes the "base price" variable array "AP(L,I)", adds a random fraction of itself, knocks off any decimal fractions with the "INT" function, then puts the resulting number, the actual local item prices, into the one-dimensional "GP(I)" array. The variable "L" is the number of the port we are in, and "I" is incremented through the loop from 0 to 5. We never did give a value to "L", but since it was never set, it automatically has a value of zero. But since zero is the number of our home-port, Hongkong, we're sitting pretty. (The Authors have used this method generally. All undimensioned variables which should start as zero aren't initialized, as both a labor- and memory-saving technique.)

We then have the "NEXT I" statement to match the "FOR I=0 TO 5" statement, and a RETURN to get us back to where we started.

Now let's go back and continue to line 50:

```

50 FOR I = 0 TO 9:
    READ LO(I):
    NEXT I:
    D = 1000:
    Y = 1860:
    GT = 1:
    C = 400:
    MW = 50:
    SH = MW:
    SR = 1:
    G = 1:
    V(0) = 1:
    GOSUB 5000:
    X$ = " ":
    GOSUB 590:

```

```
HOME:
GOTO 120
```

In line 50, we feed numbers into array "LO(I)" from DATA line 110. These values will later be processed further to give the ports we'll visit relative distances from one another.

Here's the data which is used by line 50:

```
110 DATA 21,14,7,0,35,
        49,56,42,84,200
```

We set up several other very important variables in line 50. With "D = 1000" we set the player's debt to the Triad moneylender, Elder Brother Wu, to 1000. "A thousand what?", you may ask.

There were many currencies used in the China Trade, including British pounds, Chinese Taels, Dollars Mex, and the copper Chinese "cash". For game purposes, we aren't going to specify what currency we're using. (That way we'll avoid both currency exchange problems, and the need to explain prices which might not be exactly accurate historically!) Remember, this is primarily a *game*, not an exact historical simulation. The Authors usually will choose game playability as opposed to literal accuracy in such situations.

Continuing with line 50, the next variable we have is "Y = 1860". This sets the year to 1860. "GT = 1" sets a variable which keeps track of "game time", the total supposed days elapsed during a game. "C = 400" sets the player's cash to 400 (some things). "MW = 50" sets the maximum weight of cargo that the player's vessel can carry to 50 units of cargo. "SH = MW" sets the portion of that capacity which is presently available in the ship's hold. Since we start with an empty hold, SH starts as the same as MW. "SR = 1" sets the vessel's state of repair at 100%. Whenever we print out the value of SR, we'll be converting it to percentage form. Thus an "SR" value of .10234 would print out as "10%". (Anything less than 10% state of repair, as we'll see when we encounter pirates, causes our ship to sink, and our game to unceremoniously end.) With "G = 1", we have given our craft one cannon with which to try to defend itself. "V(O) = 1", by setting a single element of the V(L) array to a value of one, tells the computer that our coming visit to Hongkong will be our first to that port o' call. The

V(L) array keeps track of how many times we've visited each port — a nice tidbit of information which the program will make available later.

We GOSUB 590 because of a peculiarity of Applesoft BASIC. Our game uses random numbers a great deal, to simulate the ebbs and flows of reality. Some microcomputers get these random numbers from "noise" generators which they otherwise use for sound effects. Other machines, including the Apple II, simulate randomness by using complex mathematical formulas to draw out pseudo-random numbers from a large built-in list of values. There is basically no problem with this second method, as long as we can't predict which number will come up next. But with the Apple II, the fact is that, every time the machine is turned on, the same list is processed the *same* way. (Other machines which use pseudo-random numbers derived from a list have a "RANDOM" or a "RANDOMIZE" statement in BASIC which "re-seeds" the random number generator to produce essentially unpredictable numbers.)

The net result of the way the Apple II handles random numbers is that it's possible to have every "random" occurrence become completely predictable to an experienced player. Prices, events, pirate attacks, etc., could happen in the same dreary order. We need some way to randomize our not-quite-random number generator. The Randomizer Subroutine at line 590 does this for Taipan. So here it is:

585 REM RANDOMIZER SUBROUTINE (590-591)

```

590 VTAB 15:
    HTAB 8:
    PRINT "PRESS <":;
    FLASH:
    PRINT "SPACEBAR":;
    NORMAL:
    PRINT "> TO START":;
    GOSUB 60:
    IF X$ = " " THEN RETURN

591 X = INT ( RND (1) * 9) + 1:
    GOTO 590

```

Our Randomizer simply cycles and recycles itself endlessly until the player presses the spacebar. Each time this line loops back upon itself, the "X=INT (RND (1) * 9) +1" statement pulls a pseudo-random number from the random number generator. Since it does this very quickly and very often, it should be impossible for a player to know how many times the subroutine has cycled. Each time it has cycled, another number in the machine's list of numbers has been used. (We had to use the statement, X\$ = " " in line 50 prior to calling the Randomizer subroutine, because had we not, there could already be a space character in X\$ before we wanted it, thus defeating our purpose. The subroutine at line 60, which will be discussed later, simply collects a character from the keyboard to put in X\$. In this case, we're looking for a "space" character.) The player's unpredictable timing in pressing the spacebar will effectively re-seed the random number generator, making certain that each game is different from the last.

Next in line 50, we use the statement, "HOME", which clears the screen. We are now finished with initialization.

Here, once again, are the lines we've used so far, so you can check your exact typing:

5 REM INITIALIZATION (10-50)

10 HOME:

A\$ = " ";

W\$ = "ELDER BROTHER WU":

LY\$ = "LI YUEN":

YS\$ = "YAMATO & SMYTHE":

TC\$ = "O, S, T, A, P, OR R"

11 B\$ = "

(Note that there are 40 spaces within the quotes which define A\$, and 39 spaces in the quotes of B\$, in line 11.)

```
20 VTAB 4:  
INVERSE:  
PRINT A$:  
HTAB 13:  
PRINT "T A I P A N":  
NORMAL:  
HTAB 13:  
PRINT "_____"
```

(The last "PRINT" statement in line 20 has 17 underline ("_") characters.)

```
21 SPEED = 100:  
VTAB 9:  
HTAB 14:  
PRINT "A G A M E I N":  
PRINT TAB(15); "C O N T E X T":  
PRINT:  
PRINT TAB(14); "HAYDEN BOOK CO."
```

```
22 SPEED = 255:  
VTAB 15:  
INVERSE:  
PRINT A$:  
NORMAL
```

```
30 DIM M$(11), G$(5), AP(9,5),  
GG(5), L(9,5), GP(5), V(9):  
FOR I = 0 TO 9:  
READ L$(I):  
NEXT I:  
FOR I = 0 TO 11:  
READ M$(I):  
NEXT I:  
FOR I = 0 TO 5:
```

```
READ G$(I):
NEXT I

40 FOR I = 0 TO 9:
FOR J = 0 TO 5:
READ AP(I,J):
AP(I,J) = AP(I,J) * B^(5 - J): NEXT J,I:
GOSUB 180

50 FOR I = 0 TO 9:
READ LO(I):
NEXT I:
D = 1000:
Y = 1860:
GT = 1:
C = 400:
MW = 50:
SH = MW:
SR = 1:
G = 1:
V(0) = 1:
GOSUB 5000:
X$ = " ":
GOSUB 590:
HOME:
GOTO 120

65 REM INITIALIZATION DATA (70-110)

70 DATA HONGKONG,FOOCHOW,
SHANGHAI,NAGASAKI,MANILA,
SINGAPORE,BATAVIA,SAIGON,
CALCUTTA,LIVERPOOL
```

80 DATA JAN,FEB,MAR,APR,MAY,JUN

81 DATA JUL,AUG,SEP,OCT,NOV,DEC

90 DATA OPIUM,SILK,
TEA,ARMS,PEPPER,RICE100 DATA 8,2,2,4,5,2,8,2,1,8,4,3,
9,3,3,6,5,4,6,4,2,8,6,5,
3,3,4,7,4,4,2,4,5,4,3,5,
3,4,5,5,2,6,3,1,4,5,3,3,
1,5,6,3,4,5,3,9,9,1,9,5110 DATA 21,14,7,0,35,
49,56,42,84,200

175 REM PRICE VARIATION SUBROUTINE (180)

180 FOR I = 0 TO 5:
GP(I) = INT(AP(L,I) + (RND(1) * AP(L,I))):
NEXT I:
RETURN

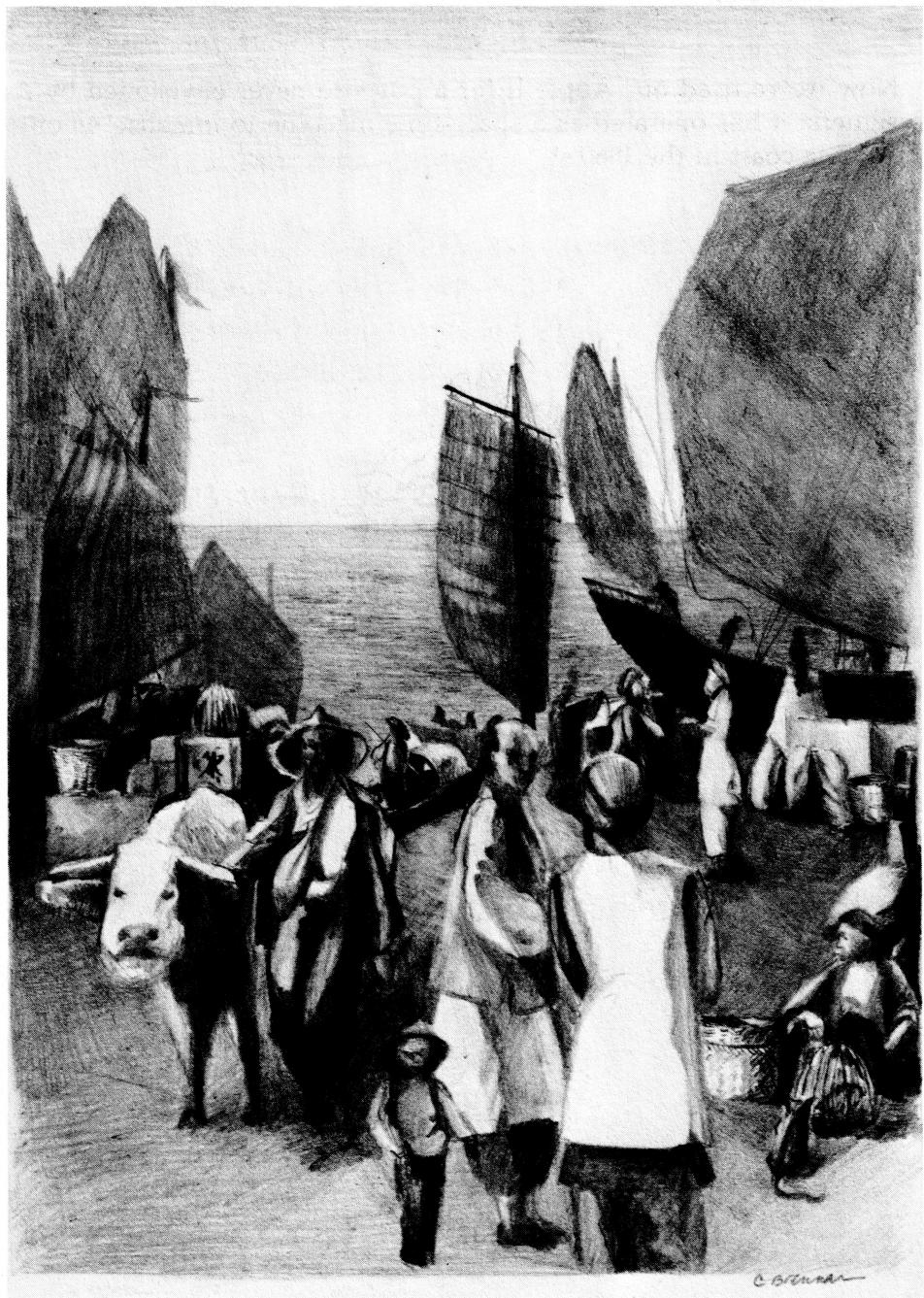
585 REM RANDOMIZER SUBROUTINE (590-591)

590 VTAB 15:
HTAB 8:
PRINT "PRESS <":
FLASH:
PRINT "SPACEBAR":
NORMAL:
PRINT "> TO START":
GOSUB 60:
IF X\$ = " " THEN RETURN

```
591 X = INT ( RND (1) * 9) + 1:  
GOTO 590
```

Now we've used our Apple II for a purpose never envisioned by its designers: it has operated as a space-time machine to initialize us onto the China coast in the 1860's!





Waiting for the ship to come in

T A I P A N

A Historical Adventure for the Apple® Computer

To Market, To Market

CHAPTER SIX

The Main Display routine begins:

```
115 REM    MAIN DISPLAY (120-150)  
  
120 GOSUB 130:  
      GOTO 220
```

With those simple instructions, we tell our Apple II to show our Main Display ('GOSUB 130'), then, with "GOTO 220", to show the local market prices and our trading menu.

Let's continue with our Main Display subroutine first:

```
130 VTAB 1:  
      HTAB 1:  
      PRINT "PORT ";L$(L);:  
      HTAB 28:  
      PRINT M$(M);". ";DA + 1;".";
```

This line does the following:

First, the cursor is positioned at the upper left-hand corner of the screen, then it prints the word, "PORT", followed by a space and then the name of the port itself. (Remember the "L\$(L)" array?) Farther to the right on the first line, at horizontal position 28, is printed the abbreviation of the month. (Originally the value of the month variable, M, will be "O", so "M\$(M)" will read "JAN".) We then print a period to neaten the abbreviation, and follow this with a space. Next, the day of the month is printed. Note that the variable for the day of the month is DA, to which we add the number 1 before printing it. This is done because the computer will actually be counting the days of the month from 0 to 29. This will appear to the player as days 1 to 30. Of course, there are not thirty days to each month, but this seems a reasonable way of doing things, considering the memory-eating alternative of setting up an exact calendar! (Once more, "gaming" takes precedence over "realism".) Finally line 130 prints the year, variable Y. (We previously used Y to set the game's beginning year to 1860.)

Continuing our Main Display subroutine, we have lines 140 and 141:

```
140 VTAB 2:
    INVERSE:
    PRINT "CASH ";
    Q = C:
    GOSUB 1330:
    NORMAL:
    VTAB 2:
    HTAB 28:
    PRINT "GUNS ";
    VTAB 3:
    PRINT "DEBT ";
    Q = D:
    GOSUB 1330:
    VTAB 3:
    HTAB 28:
    PRINT "HOLD ";
    Q = SH:
```

GOSUB 1330

141 VTAB 4:

INVERSE:

**PRINT "GOODS ABOARD SHIP HONGKONG GODOWN":
NORMAL**

A player will often find that quantities in *Taipan* (such as cash, debt, and trade goods) will become enormously large. Applesoft BASIC is designed so that it will allow the display of numbers under one billion (1,000,000,000) in a normal fashion, but it will display numbers from one billion on (including negative numbers) in a computer version of "scientific notation".

This display of big numbers is a bit different than the usual sort of scientific notation. In the non-computer type of scientific notation which many of us have encountered in school, one billion would look like this:

1 x 10⁹

This could be written out in words as "one times ten to the ninth power". But one billion, as displayed by our Apple II, would normally look like:

1E+09

Not too swift, actually. Not something which a player who is not familiar with computers would be likely to understand! (Even the non-computer style of showing scientific notation is hardly neat or convenient for the layperson to comprehend — the computer version is just worse.)

Yet even using an awkward version of scientific notation, numbers may simply get too long to display in the number of spaces we have available for them in the 40-column format we're using for the Apple II. So, as a solution for these problems, we've set up a special Big Number subroutine:

1325 REM BIG NUMBER SUBROUTINE (1330-1370)

1330 IF ABS (Q) < 1E6 THEN PRINT INT (Q)::

NORMAL:

PRINT " ":

RETURN

1331 IF ABS (Q) < 1E9 THEN Q = Q / 1E6::

Q\$ = "MIL":

GOTO 1335

1332 IF ABS (Q) < 1E12 THEN Q = Q / 1E9::

Q\$ = "BIL":

GOTO 1335

1333 IF ABS (Q) > = 1E12 THEN Q = Q / 1E12::

Q\$ = "TRL":

GOTO 1335

1335 PRINT INT (Q)::Q\$::

NORMAL:

PRINT " ":

1337 RETURN

1340 FOR I = 12 TO 18

1345 HTAB 1

1350 VTAB I:

PRINT A\$::

1360 NEXT I

1365 PRINT:

VTAB 12

1370 RETURN

Look back at line 140. There we set the value of variable Q to equal C (our cash). We did this just before GOSUBbing 1330. The Big Number subroutine takes this value held temporarily in Q and, if the number is less than one million (or negative one million), simply prints it in neatly formatted style.

If the number is a positive or negative number of at least one million and less than a billion, the number is formatted so that . . .

1,242,144,495

. . . would come out looking like:

124.2MIL.

Numbers in the billions and trillions are handled in a similar manner by the Big Number subroutine. The abbreviations "MIL.", "BIL." or "TRL." are shown in reverse video to catch the player's eye.

Negative numbers will be indicated by a minus sign (-) to their left.

The way the Big Number subroutine handles astronomical-sized quantities gives the player meaningful information without making havoc of our display with numbers which get too long for the space we have available. We use the Big Number subroutine three times in line 140 alone, and we'll be calling on it to help us with other potentially huge numbers whenever we need to.

So, in line 140 we've printed our cash (originally four hundred), the number of guns with which our vessel is equipped (one for starters), our debt to Elder Brother Wu (one thousand as the game begins), and the available hold capacity of our vessel (fifty thus far). Line 141 next displays a bright bar across the screen, with the words, "GOODS", "ABOARD SHIP", and "HONGKONG GODOWN" in reverse video embedded in the bar. (Hongkong is where our only godown — warehouse — is located.)

The last line of our Main Display subroutine is:

```

150 FOR I = 0 TO 5:
    VTAB 5 + I:
    PRINT G$(I):
    VTAB 5 + I:
    HTAB 11:
    PRINT CHR$(133):::
    Q = SG(I):
    GOSUB 1330:
    VTAB 5 + I:
    HTAB 26:
    PRINT CHR$(133):::
    Q = GG(I):
    GOSUB 1330:
    NEXT I:
    INVERSE:
    PRINT A$:
    NORMAL:
    RETURN

```

This line displays the quantities of goods we have aboard ship and in our Hongkong godown.

We start a loop, incrementing from 0 to 5. Within the loop, we first "VTAB 5+I", and then "PRINT G\$(I)". This method allows us to show G\$(I), the names of the trade items, in neat columns, one item name to a line. Then another VTAB and an "HTAB 11" put our cursor eleven columns to the right, so that it displays vertical bars, CHR\$(133), to the right of the item names. Next, the first call of the Big Number subroutine (GOSUB 1330) formats the numbers of items aboard ship. Then we go over a few more columns with another VTAB and HTAB, and display another vertical bar, with the Big Number subroutine now formatting the numbers for the amount of each item stored in our godown. After we've displayed information on all our goods aboard ship or stored, the NEXT I ends the loop. We then do an INVERSE, and print a thick bright horizontal bar (A\$) on the next line. Last, we execute NORMAL, and RETURN from the subroutine.

Okay. Now we've finished our Main Display, and have RETURNed to line 120. Line 120 immediately sends us away with "GOTO 220". (This time we seem to have outstayed our welcome — we're kicked off the premises without so much as a "y'all come back, hear?")

Lines 220 and 221 start our Market routine:

212 REM MARKET (220-350)

217 REM MARKET PRICES (220-221)

220 GOSUB 790:

 GOSUB 1340:

 VTAB 11:

 INVERSE:

 HTAB 8:

 PRINT " ";L\$(L);" MARKET PRICES ":

 NORMAL:

 PRINT A\$:

 FOR I = 0 TO 4 STEP 2:

 VTAB 13 + I / 2:

 HTAB 1:

 PRINT G\$(I);:

 HTAB 10:

 PRINT GP(I);:

 HTAB 21:

 PRINT G\$(I + 1);

221 HTAB 30:

 PRINT GP(I + 1);

 NEXT I

What have we done here? First, we called the Events subroutine which starts at 790. The Events subroutine is used to present a variety of happenings and opportunities which may come up from time to time. Since we will discuss the Events subroutine in a later chapter, let's for the

moment assume that we've already RETURNed from there. After the GOSUB, line 220 displays the thick bright bar, A\$, then name of the port, followed on the same line by the words, "MARKET PRICES".

A loop is then started, with the variable "I" being incremented from 0 to 5. Within the loop, we print two item names and their formatted prices on each of three lines. (Note the fancy footwork with HTAB and VTAB values. These allow us to keep things neat on the screen.)

Line 230 of our Market routine is simple:

```
225 REM MARKET MENU
230 PRINT:
      VTAB 18:
      PRINT " B)UY, S)ELL, L)EAVE, OR R)ETIRE?"
```

Here we have our Market Menu. We emphasize the first letters of each choice, because the player will choose by pressing the appropriate letter.

Lines 240 to 244 are the Market Menu Input routine, and actually handle the player's choices.

```
235 REM MARKET MENU INPUT (240-244)
240 GOSUB 60:
      IF X$ = "B" THEN T$ = "BUY":
      X = 1
241 IF X$ = "S" THEN T$ = "SELL":
      X = 2
242 IF X$ = "L" THEN 360
243 IF X$ = "R" THEN 1300
```

```
244 IF X$ <> "B" AND X$ <> "S" AND X$ <> "L" AND X$ <>
"R" THEN GO SUB 770:
GOTO 240
```

Since line 240 needs our Get String subroutine at 60 to work, let's examine that, too:

```
55 REM GET$ SUBROUTINE (60-64)

60 POKE - 16368,0

61 IF PEEK ( - 16384) < 128 THEN 61

62 X$ = CHR$ ( PEEK ( - 16384) - 128)

63 POKE - 16368,0

64 RETURN
```

A reasonable person might ask why we need the Get String subroutine to handle simple input, rather than just using an INPUT statement. The main reason we're using this subroutine is that we are thus able to "screen out" undesired input.

The most common form of undesired input is the accidental input of options. Sometimes even veteran players will become inattentive and will press a key which is incorrect. (One possibility is that the player will make an option choice which is legitimate only as an option in another menu.)

The Market Menu Input routine takes the player's input and checks it for legitimacy. If the input is not proper, the program routes to a subroutine at line 770, where a short beep is sounded to make the player aware of the error.

Here's the Input Error subroutine, line 770:

```
765 REM INPUT ERROR SUBROUTINE (770)
770 PRINT "^G":  
      RETURN
```

[Note: “^G” is the “beep” character. It is typed by first pressing the “control” key, then pressing the “G” while the “control” is still down. Don’t expect it to look like “^G” on your screen listing or program print-out, however!]

On the other hand, if the input is legitimate, in other words, one of the letters indicted as options in the Market Menu (B, S, L or R), the routine then processes the input and routes the program’s flow in order to make the right things happen in response.

Now let’s go into the Market Menu Input routine and the Get String subroutine in detail.

Line 240 immediately GOSUBs to the Get String subroutine at 60. Our Get String subroutine is one of our most-used routines in Taipan. First, line 60 empties the contents of the keyboard buffer (POKE – 16386). This clears any keystrokes from your Apple II, and ensures that any previous keyboard input isn’t confused with new input. (The Get String subroutine will look at the last character typed.) Next, in line 61, the statement “IF PEEK (– 16384,) < 128 THEN 61” is run. This causes the program to loop back to the beginning of the same line unless a key is pressed on the keyboard. Then line 62 sets the contents of X\$ to equal the character in the keyboard buffer: “X\$ = CHR\$ (PEEK (– 16384) – 128)”. This makes the contents of X\$ equal to whatever key on the keyboard has been pressed last.

The Get String subroutine checks to see if any keys have actually been pressed. If not, line 61 is restarted at its beginning. But if a key has been pressed, we RETURN to line 240, with X\$ holding the character corresponding to the key which the player pressed.

Now lines 240 to 244 have to examine the contents of X\$, and make something happen. Here’s how:

If X\$ holds the letter “B” (for Buy), then line 240 puts the word, “BUY” into T\$, and sets the numerical variable, X, to a value of 1.

(We're using X in this routine as a so-called "flag". A variable is considered a flag when it is used later in a program to indicate something which happened previously. Here, "X=1" means "buy". We don't want to use X\$ as a flag, because we'll soon use it to hold other characters.)

But if X\$ doesn't hold a "B", line 241 continues to check what it might hold. If X\$ contains an "S" (for Sell), then T\$ is loaded with the word, "SELL", and the "X" flag is set to 2.

But that might not be the case, either — after all, there are lots of keys on a keyboard! So line 242 keeps checking. If the contents of X\$ is the letter "L" (for Leave), the program jumps to line 360, where the Other Options routine is located. (We'll get to that routine, also, in a later chapter — trust us.)

If we pressed a key other than B, S or L, line 243 checks to see if the key we pushed is an R (for Retire). If so, the program flow jumps to the That's All Folks routine at line 1300.

But what if we had pressed none of the keys covered by this routine (neither the B, S, L or the R key)? In that case, line 244 has the computer GOSUB to line 770, the Input Error subroutine, where a short routine plays that beep tone we mentioned earlier. Then the computer starts out at the beginning of line 240 all over again, hoping upon hope that the next key pressed will make sense to it.

In only two cases, the program will simply drop through to line 250, the start of the Buy/Sell routine. That will, naturally, happen only if a B or an S is pressed.

The Buy/Sell routine allows items to be bought and sold. (What else did you expect?) It handles choices of trade goods, handles the input of the amounts to be bought or sold, checks whether the player has sufficient cash when buying, or sufficient numbers of goods to be sold. The Buy/Sell routine checks for almost any kind of error in input, and prevents certain kinds of "cheating" by the player. It uses a special number input subroutine which resists scrolling and allows the player to buy "all" he or she can afford of an item by simply pressing the letter "A", or sell "all" by pressing the same key. After each transaction, the Main Display subroutine is run in order to update the amounts of trade goods held by the player, and the player's cash on hand. Also after every purchase or sale, the player is returned to the Market Menu and Market Menu Input routines. The Buy/Sell routine is designed to be compact. To use as little memory as possible, it makes multiple use of as many lines as

possible. There are lines, for instance, which handle either the buying or the selling of any of the six items of trade, while other lines handle other buying or selling functions.

This is the start of the Buy/Sell routine:

245 REM BUY/SELL (250-351)

247 REM ITEM MENU (250)

250 VTAB 18:
 PRINT A\$;
 VTAB 18:
 HTAB 3:
 PRINT T\$;" ";TC\$;"?":
 GOSUB 260:
 GOTO 280

If you recall (and even if you don't), A\$ is a line of 40 spaces. We print it at position VTAB 18 on the screen, in order to clear what we had there before, the Market Menu. Next, using HTAB 2 to move over two columns, we print T\$, which we've just made to contain either the word "BUY" or the word "SELL". After T\$, we display a space, followed by TC\$, then a question mark. We previously put the first letters of each of our trade items into TC\$, back in line 10 during initialization. So what we get on the screen is either . . .

BUY O, S, T, A, P OR R?

or . . .

SELL O, S, T, A, P OR R?

Since we don't have nearly enough room to spell this prompt out in whole words, like "SELL OPIUM, SILK, TEA, ARMS, PEPPER OR RICE?", and those words are written right above in the Market Prices, the letters will suffice.

Just as we did in the Market Menu Input routine, we now have to take, and process, the player's input. So lines 260 through 270 will be our Item Choice subroutine. Since we'll have other places in the program where a choice of items will be needed, we want to make this a subroutine, rather than having similar lines scattered throughout the program. For the same reason, it's also useful to make the Item Choice subroutine as generalized, and therefore as flexible, as possible. (It often seems that the realities of memory-conscious programming conspire against "elegant", top-down programming.)

Here is our Item Choice subroutine:

```

255 REM      ITEM CHOICE SUBROUTINE (260-270)

260 GOSUB 60:
    IF X$ = "0" THEN X1 = 0

261 IF X$ = "S" THEN X1 = 1

262 IF X$ = "A" THEN X1 = 3

263 IF X$ = "P" THEN X1 = 4

265 IF X$ = "R" THEN X1 = 5

266 IF X$ <> "0" AND X$ <> "S" AND X$ <> "T" AND X$ <>
    "A" AND X$ <> "P" AND X$ <> "R" THEN GOSUB 770:
    GOTO 260

270 RETURN

```

Our Item Choice subroutine first GOSUBs to the Get String Subroutine at 60, and gets an input from the keyboard, just like in our Market Menu Input routine at 240. It then sets the variable X1 as a flag to identify the chosen item. If an invalid key is pressed, the program GOSUBs line 770, the Input Error subroutine, to signal the player to "watch it, buster!" Much like lines 240 through 244, isn't it?

Here's a review of the lines we've used in this chapter:

```
55 REM GET$ SUBROUTINE (60-64)
60 POKE - 16368,0
61 IF PEEK ( - 16384) < 128 THEN 61
62 X$ = CHR$ ( PEEK ( - 16384) - 128)
63 POKE - 16368,0
64 RETURN
115 REM MAIN DISPLAY (120-150)
120 GOSUB 130:
      GOTO 220
130 VTAB 1:
      HTAB 1:
      PRINT "PORT ";L$(L)::
      HTAB 28:
      PRINT M$(M)::. ";DA + 1";"Y
140 VTAB 2:
      INVERSE:
      PRINT "CASH ";
      Q = C:
      GOSUB 1330:
      NORMAL:
      VTAB 2:
      HTAB 28:
      PRINT "GUNS ";G:
```

```
VTAB 3:
PRINT "DEBT ";
Q = D:
GOSUB 1330:
VTAB 3:
HTAB 28:
PRINT "HOLD ";
Q = SH:
GOSUB 1330

141 VTAB 4:
INVERSE:
PRINT "GOODS ABOARD SHIP HONGKONG GODOWN":
NORMAL

150 FOR I = 0 TO 5:
VTAB 5 + I:
PRINT G$(I):
VTAB 5 + I:
HTAB 11:
PRINT CHR$(133):
Q = SG(I):
GOSUB 1330:
VTAB 5 + I:
HTAB 26:
PRINT CHR$(133):
Q = GG(I):
GOSUB 1330:
NEXT I:
INVERSE:
PRINT A$:
NORMAL:
RETURN
```

212 REM MARKET (220-350)

217 REM MARKET PRICES (220-221)

220 GOSUB 790:

GOSUB 1340:

VTAB 11:

INVERSE:

HTAB 8:

PRINT " ";L\$(L);" MARKET PRICES ":

NORMAL:

PRINT A\$:

FOR I = 0 TO 4 STEP 2:

VTAB 13 + I / 2:

HTAB 1:

PRINT G\$(I);:

HTAB 10:

PRINT GP(I);:

HTAB 21:

PRINT G\$(I + 1);

221 HTAB 30:

PRINT GP(I + 1):

NEXT I

225 REM MARKET MENU (230-244)

230 PRINT:

VTAB 18:

PRINT " BUY, SELL, LEAVE, OR RETIRE?"

235 REM MARKET MENU INPUT (240-244)

```
240 GOSUB 60:
  IF X$ = "B" THEN T$ = "BUY":
  X = 1

241 IF X$ = "S" THEN T$ = "SELL":
  X = 2

242 IF X$ = "L" THEN 360

243 IF X$ = "R" THEN 1300

244 IF X$ <> "B" AND X$ <> "S" AND X$ <> "L" AND X$ <> "R" THEN GOSUB 770:
  GOTO 240

247 REM      ITEM MENU (250)

250 VTAB 18:
  PRINT A$;;
  VTAB 18:
  HTAB 3:
  PRINT T$;" ";TC$;"?":
  GOSUB 260:
  GOTO 280

255 REM      ITEM CHOICE SUBROUTINE (260-270)

260 GOSUB 60:
  IF X$ = "O" THEN X1 = 0

261 IF X$ = "S" THEN X1 = 1
```

```
262 IF X$ = "A" THEN X1 = 3
263 IF X$ = "P" THEN X1 = 4
265 IF X$ = "R" THEN X1 = 5
266 IF X$ <> "O" AND X$ <> "S" AND X$ <> "T" AND X$ <> "A" AND X$ <> "P" AND X$ <> "R" THEN GOSUB 770:
    GOTO 260
270 RETURN
765 REM INPUT ERROR SUBROUTINE (770)
770 PRINT "^G";:
    RETURN
```

[Note: “^G” is the “beep” character. It is typed by first pressing the “control” key, then pressing the “G” while the “control” is still down. Don’t expect it to look like “^G” on your screen listing or program print-out, however!]

```
1325 REM BIG NUMBER SUBROUTINE (1330-1370)
1330 IF ABS (Q) < 1E6 THEN PRINT INT (Q);:
    NORMAL:
    PRINT "      ";
    RETURN
1331 IF ABS (Q) < 1E9 THEN Q = Q / 1E6:
    Q$ = "MIL":
    GOTO 1335
```

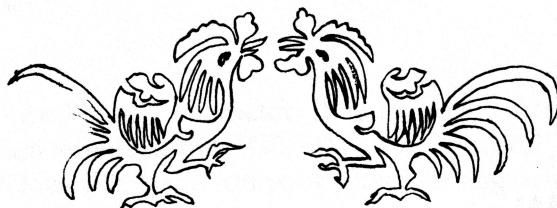
```
1332 IF ABS (Q) < 1E12 THEN Q = Q / 1E9: Q$ = "BIL":  
    GOTO 1335  
1333 IF ABS (Q) > = 1E12 THEN Q = Q / 1E12: Q$ = "TRL":  
    GOTO 1335  
1335 PRINT INT (Q);Q$;:  
    NORMAL:  
    PRINT " "  
  
1337 RETURN  
  
1340 FOR I = 12 TO 18  
  
1345 HTAB 1  
  
1350 VTAB I:  
    PRINT A$;  
  
1360 NEXT I  
  
1365 PRINT:  
    VTAB 12  
  
1370 RETURN
```

So, to sum up what we've done in this chapter:

Line 120 sent the program to the Main Display subroutine, starting at 130. The Main Display subroutine (lines 130 to 150) displays a whole mess

of important information on the screen, like where we are, what we've got with us, what day it is, etc.

After RETURNing from the Main Display subroutine, line 120 sends us to the Market routine starting at line 220. The first portion of the Market routine is the Market Price routine, which first GOSUBs to the Events subroutine at 790, then displays how much each item costs. Continuing the Market routine, line 230 is our market Menu routine, displaying options. Lines 240 to 244 are the Market Menu Input routine. They handle the player's choices, working with the Get String subroutine at line 60, and the Input Error subroutine at line 770. With correct input, the program then goes to the lines covered in the next chapter, which are at the heart of our game, lines which handle the trading process itself.







Canton—Commissioner Lin burns opium

T A I P A N

A Historical Adventure for the Apple® Computer

Cash on the Barrelhead

CHAPTER SEVEN

At the end of the last chapter we either told the computer that we wanted to buy, or that we wished to sell. We also chose the item we wished to deal with. The Market routine continued after the Item Choice subroutine (lines 260 through 270) RETURNed program flow to the end of line 250 (the first line of the Buy/Sell routine). Line 250 finally sent us away with GOTO 280.

Here's line 280:

```
280 IF X=1 AND GP(X1) > C THEN VTAB 18:  
PRINT "YOU CAN'T AFFORD ANY "G$(X1)". . .":  
GOSUB 760:  
GOTO 230
```

Line 280 first checks to see if two things are true: that we're both trying to buy an item *and* that we don't have enough cash to buy a single unit of that item. If both of these things are true, we are told that we can't afford any of this item, we GOSUB 760 to play a note to underscore our error, and we RETURN to our Market Menu at 230.

Thus line 280 only operates if we're trying to buy something we can't afford.

If you were to walk into a real estate office, a smiling agent might ask whether you wanted to buy or sell property.

You might reply: "I'd like to buy some real estate, I do believe, if it is not too much trouble."

"No trouble at all. I live to serve," says the agent, wringing his/her hands in anticipation of commission points. The agent gestures at shoeboxes full of real estate deeds. (Each is marked with a single initial letter.)

The agent says, "We handle the finest in Shanties, Cabins, Bungalows, Houses, Mansions and Palaces. In which category are you interested?"

You answer: "Uh, I think your 'Palace' category would suit me fine today, thank you so very much, please."

Now, at this point it is an exceedingly remote possibility that the real estate agent would just grab a boxful of deeds marked with a "P", and ask, "And exactly how many Palaces would you like?"

It's much more likely that the agent will attempt to "qualify" you. In other words, the agent will question you in depth, trying to determine whether you can afford even a single Palace.

That's what line 280 does. It "qualifies" the player for the purchase.

Line 290 does a similar check:

```
290 IF X = 2 AND SG(X1) < 1 THEN VTAB 18:
  PRINT "YOU HAVE NO "G$(X1)" ABOARD!      ";
  GOSUB 760:
  GOTO 230
```

All line 290 does is to check whether you're trying to sell something you don't have. (Keep that deed for the Brooklyn Bridge in your pocket!)

Line 760, the No-Can-Do subroutine, both plays a note and calls the Delay subroutine, starting at line 780. Here's both of those subroutines:

```
755 REM NO CAN DO SUBROUTINE (760)
```

```
760 PRINT "AG":  
GOSUB 780:  
RETURN
```

[Note the "AG" in line 760. Remember, it is typed in by first pressing the Control key, then pressing the "G" key while the Control is still pressed.]

```
775 REM DELAY SUBROUTINE (780-783)
```

```
780 POKE - 16384,0
```

```
781 FOR I = 0 TO 250
```

```
782 IF PEEK (- 16384) > 127 THEN  
POKE - 16384,0:  
RETURN
```

```
783 NEXT I:  
RETURN
```

Actually, line 760 (our No-Can-Do subroutine) is much like the Input Error subroutine we already introduced, line 770. The difference is that in line 760, we also use the Delay subroutine at lines 780 to 783.

The Delay subroutine is a handy and "user-friendly" little scrap of programming. It's used to purposely slow down the computer whenever we are giving the player a brief message. It gives the player plenty of time to read the message. But since some folks read faster than others, the Delay subroutine gives the player the option of simply pressing the spacebar of the computer in order to end the message. This can considerably speed play, especially when the player becomes familiar with some of the more common brief messages. Notice that a "FOR...TO...NEXT" loop is used just to eat up time, while constantly checking to see if the spacebar is pressed. If either the variable I becomes incremented to 250 or the spacebar is pressed, the subroutine RETURNS.

The way that the program detects the spacebar is with PEEK and POKE statements. A full explanation would be very technical, but the nitty gritty is that these statements clear out the keyboard buffer, then look for input. When the spacebar is pressed, line 782 RETURNS us from the subroutine, even if the allotted time hasn't run out.

With lines 280 and 290 we screened out both too-poor buyers and empty-handed sellers. Now that we've banished the riff-raff, we can really start doing business with our next line, 300:

```
300  VTAB 18:
      PRINT A$;;
      VTAB 18:
      PRINT T$;" HOW MUCH ";G$(X1);:
      PRINT "? ";:
      NUM$ = "":
      NUM = 0:
      GOSUB 310:
      GOTO 320
```

You'll recall that we put either "BUY" or "SELL" into T\$ back in lines 240 through 244, our Market Menu Input routine, and that X1 is used as a flag to indicate the type of item we're dealing with. Thus, line 300 can ask any of twelve questions. It can ask the player how much of any one of six items should be either bought or sold.

Next, the line makes sure that the string variable, NUM\$, is empty. This is important, because we'll be using NUM\$ to collect keyboard input in a moment. Likewise, the numerical variable, NUM, is set to zero.

Line 300 then GOSUBs to 310, where we start perhaps the trickiest routine in the game, the Sooper Dooper Number Scooper subroutine:

```
305  REM  SOOPER DOOPER NUMBER SCOOPER SUBROUTINE
      (310-314)
310  PRINT CHR$(8);:
```

```

INVERSE:
PRINT " ";
NORMAL:
GOSUB 60:
IF LEN (NUM$) > 0 AND ASC (X$) = 8 THEN PRINT X$;;
PRINT" ";
PRINT X$;X$;;
PRINT" ";
FG=1:
IF LEN (NUM$) = 1 THEN NUM$ = " ":
FG = 0:
GOTO 310

311 IF FG = 1 THEN NUM$ = LEFT$ (NUM$, LEN (NUM$) -
1):
FG = 0:
GOTO 310

312 IF ASC (X$) = 65 OR ASC (X$) = 13 THEN RETURN

313 IF ASC (X$) < 48 OR ASC (X$) > 57 THEN 310

314 NUM$ = NUM$ + X$:
PRINT CHR$ (8);X$;;
INVERSE:
PRINT " ";
NORMAL:
GOTO 310

```

Looks nasty, huh? Actually, it's fairly simple. Like filling out tax return forms is "simple", that is . . . But however complex it might seem, it simplifies input for the player. If the Delay subroutine was "user-friendly", the Sooper Dooper Number Scooper is downright fresh!

Let's first explain what it is we want this subroutine to do:

We need a routine which will allow us to enter numbers into the computer. We'd like to be able to have a special key, the letter A, which will allow us either to buy all of an item which we can afford, or to sell all of an item which we have aboard ship.

Further, we'd like to filter out any accidental input of non-numeric keys (except "A"), and especially to prevent the players from putting a minus sign in front of their numbers. (This could be a form of "cheating". A player could "buy" a *negative* amount of a item, thus actually *getting* cash and making the player's ship *lighter!*)

Other things it'd be nice to do are to have the numbers input by the player print on the screen, and to allow the player to "back up" using the "back-arrow" key to correct mistakes. Let's also throw in a cursor, since folks expect to see one of those things. Finally, we'd like to get out of the Sooper Dooper Number Scooper subroutine either by pressing "RETURN", or by using the "A" key.

We've crammed features for handling all of this wild and crazy stuff into this routine.

Yes, it was hard to write — but not nearly as difficult as explaining it. Here's our attempt:

First, we have PRINT CHR\$ (8). CHR\$ (8) is the "backspace" character. So, wherever we've positioned the invisible cursor prior to using this subroutine for number input, it's moved back one space. A bright block (an INVERSE space: '■') is then displayed as a visible "artificial" cursor. Notice that we leave a semicolon (;) after the cursor, so the computer will print its next character on the same line, just after the cursor, rather than on the next line. (The computer can be made not to display its own cursor, but it always has at least an invisible cursor it keeps track of, so it knows where to PRINT next.) After displaying the INVERSE-space cursor, we NORMALize the display.

The next thing line 310 does is to GOSUB line 60. That is our Get String subroutine, which we earlier explained. It's enough for now to remember that line 60 puts characters from the keyboard into X\$.

When a key is pressed, the Get String Subroutine RETURNS us to line 310. Now we have the statements: IF LEN (NUM\$) > 0 AND ASC (X\$)=8 THEN PRINT X\$;: PRINT " ";: PRINT X\$;X\$;:. This part of the Sooper Dooper Number Scooper subroutine allows the player to back-

space the cursor to delete any number that might have been typed by mistake. It first checks that NUM\$ (which holds the accumulated characters from the keyboard in string, rather than numerical form), has a LENGTH greater than zero — that it has something in it, in other words. This check is done because otherwise we might backspace too far to the left. Since this portion of the line will operate only if numbers are already displayed and a back-arrow is pressed, X\$ would at this point naturally contain a back-arrow, or backspace, character. We have to print a space character (" ") next, to delete our "artificial" cursor. We then have to backspace twice more (PRINT X\$;X\$;), in order to put our "artificial cursor" just to the left of the digit we want to erase in the first place. Next, we have to print another space, to actually delete that digit.

Okay. So we've been able to delete an unwanted digit from the screen. But we've still got to get it out of the computer's memory. We then set a flag variable to a value of one (FG = 1). (We use this flag to remind the program that we're shortening the length of NUM\$.) The next statement is: IF LEN (NUM\$)=1 THEN NUM\$ = " ". That statement simply says that if the length of the string, "NUM\$" is one character, then we'll make it zero characters long. So now what we see is really what we get. Next, we reset the flag, FG, to zero, it having served its purpose. And finally, having corrected the last mis-typed digit, we GOTO the top of line 310, so we can begin typing in new characters.

Line 311 works much like the last three statements in line 310, except it operates in a situation where there is already more than one digit contained in NUM\$. NUM\$ will have its right-most character cut off. In other words, the last character entered will be nipped out of NUM\$.

Got that? If so, please explain it to us. (Just kidding, folks!)

If we haven't pressed the back-arrow, the next part of the Sooper Dooper Number Scooper subroutine, line 312, checks to see if we've pressed either the "A" (character 65) or the "RETURN" key (character 13). This statement is the only way out of the subroutine. If either of these keys were pressed, program flow RETURNS from the subroutine to where it came from — in this case, line 300. (Afterward, as we'll soon show, line 300 sends us to routines starting at 330, where either the numbers held in string form in NUM\$, or that "A" get processed further.)

If we've pressed neither "back-arrow," "A," nor "RETURN," the next part of the routine, line 313, screens the input to make certain that we've at least pressed one of the number keys: "IF ASC (X\$) < 48 OR ASC (X\$) > 57 THEN 310". Since the ASCII value of the characters for the num-

bers zero through nine run from 48 to 57, this line simply routes us back to square one, restarting the line at the top, if any non-numeric keys (other than "back-arrow", "A" or "RETURN", which were already taken care of) were pressed.

Now, in line 314, we see how NUM\$ gets those number characters into it: "NUM\$ = NUM\$ + X\$". This adds the last number character from the keyboard (X\$) to the right end of the string contained in NUM\$. Since this is the only way anything can get into NUM\$, NUM\$ will only hold number characters.

After that, we have "PRINT CHR\$ (8);X\$;". This uses the backspace character (8) to erase the existing "artificial" cursor, then prints X\$, which will display a numerical digit in the old cursor's place. Then we have another INVERSE space to serve as a cursor, and NORMAL to make screen printing, well . . . normal.

Finally, we have a "GOTO 310", which takes us to the top of line 310, so the player can add more digits to the number or finish by pressing "RETURN".

That's the Sooper Dooper Number Scooper.

Let's assume the player has pressed either "A" or "RETURN", and we're back at line 300.

Following the "GOSUB 310" in line 300, is "GOTO 320". Lines 320 through 335 continue with gathering the quantity to be bought or sold. Let's look at 'em:

```

320 IF X = 1 THEN IF X$ = "A" THEN PRINT CHR$(8);:
    INVERSE:
    PRINT "ALL";:
    NUM = INT (C / GP(X1))
    GOTO 321

321 IF X = 1 AND X$ <> "A" THEN NUM = VAL (NUM$)

330 IF X = 2 THEN IF X$ = "A" THEN PRINT CHR$(8);:
    INVERSE:
    PRINT "ALL";:
    NUM = SG(X1)
    GOTO 321

```

```
331 IF X = 2 AND X$ <> "A" THEN NUM = VAL (NUM$)
```

```
335 NORMAL
```

Line 320 through 335 convert the output of the Sooper Dooper Number Scooper subroutine into quantities of goods to be bought. The first part of line 320 checks if we are buying ($X=1$), and whether the last key we pressed is an "A". If both these things are true, it backspaces the invisible cursor, then briefly prints the word "ALL" in inverse video. The last portion of line 320, "NUM=INT (C / GP(X1))", sets the value of the numerical variable NUM to equal the largest affordable whole number of the item the player wishes to buy. It does this by dividing the player's cash by the unit price of the chosen item and then using the INT function to round off the result to a whole number. (Throughout Taipan, we'll force the computer to nearly always output quantities as whole numbers, and to always accept only whole numbers as inputs. This neatens the screen, simplifies play, and does nothing to hurt "realism".)

Line 321, "IF X=1 AND X\$ <> "A" THEN NUM=VAL (NUM\$)", operates when we are buying and we have not pressed the "A" key. It converts the string contents of NUM\$ into a quantity to be stored in numerical variable NUM.

A similar process goes on in lines 330 and 331, but only if we're selling cargo items. The first part of line 330 works just like the beginning of line 320, so we'll skip over that. The last part, which operates if the player has pressed "A", is "NUM=SG (X1)". This puts a number equal to all of our shipboard quantity of the chosen item into the variable NUM.

Line 331 operates if we're selling, and have input numbers (instead of pressing "A") via the Sooper Dooper Number Scooper. This works just like a line 321, converting numbers in the string NUM\$ to the variable, NUM. Finally, we use NORMAL to stop displaying in INVERSE mode.

This ends our Transaction Quantity routine.

After lines 320 through 330, we now have the quantity of goods to be bought or sold, stored in NUM. Lines 340 to 341, and 350 to 351 of our Market routine respectively finish the transaction of buying or selling items. Here are those last four lines of the Market routine:

```
340 IF X = 1 THEN IF NUM * GP(X1) > C THEN PRINT:  
    VTAB 18:  
    PRINT "YOU CAN'T AFFORD SO MUCH! ";:  
    GOSUB 760:  
    VTAB 18:  
    PRINT A$;:  
    GOTO 300  
  
341 IF X = 1 AND NUM * GP(X1) < = C  
    THEN SG(X1) = SG(X1) + NUM:  
    SH = SH - NUM:  
    C = C - GP(X1) * NUM:  
    GOSUB 130:  
    GOTO 230  
  
350 IF NUM > SG(X1) THEN VTAB 18:  
    PRINT A$;:  
    VTAB 18:  
    PRINT "YOU DON'T HAVE THAT MUCH! ";:  
    GOSUB 760:  
    VTAB 18:  
    PRINT A$;:  
    GOTO 300  
  
351 SG(X1) = SG(X1) - NUM:  
    SH = SH + NUM:  
    C = C + (NUM * GP(X1)):  
    GOSUB 130:  
    GOTO 230
```

Line 340 first makes certain we're buying. If we're selling cargo, the rest of the line is skipped. (A similar check would prevent line 341 from operating.) It then checks if we're ordering more than we can pay for. If

we can't afford the quantity we've input, it tells us we can't afford that amount, GOSUBs to the No-Can-Do subroutine at 760, clears the message it's just given us using A\$ (a line of 31 spaces), and goes back to line 300 to ask us once again how much we want to buy.

But if we're buying and *can* afford the quantity we've asked for, line 341 comes into play. The shipboard amount of that item, SG(X1), is increased by the quantity in the variable NUM. The available cargo space aboard ship (SH) is decreased by NUM, and the player's cash (C) is lowered by the price of the chosen item, multiplied by the amount the player requested. Line 341 then GOSUBs 130 to update the Main Display to show how much cargo and money we now have, and finally goes back to the Market Menu and the reset of the Market Option routine, with GOTO 230.

Lines 350 and 351 are the flip side of the proposition, and handle selling in about the same way that 340 and 341 handled buying.

Since if we were buying, the program flow wouldn't have gotten this far, we don't have to check to make sure we're trying to sell. To get to line 340, we'd have to be selling something. Line 350 therefore first checks to see if we're trying to sell more of an item than we have aboard ship. If we are, it tells us we don't have that much, GOSUBs to the No-Can-Do subroutine, removes its message, then puts us back at line 300, where it can again ask us how much is to be sold.

If the player does have at least as much cargo as has been input, NUM is subtracted by line 351 from the amount of that item aboard ship, the ship's available capacity is increased accordingly, the player's cash is increased by the unit price of the item multiplied by the number sold, and the Main Display is updated. Control is returned to the Market Options routine starting at 230.

That, Dear Reader, is curtains for the Market routine, may it rest in peace. It took us two chapters, but we are finally finished.

Here's a recap of the lines we've used in this chapter:

```
280 IF X=1 AND GP(X1) > C THEN VTAB 18:
PRINT "YOU CAN'T AFFORD ANY "G$(X1)". ";
GOSUB 760:
GOTO 230
```

```
290 IF X = 2 AND SG(X1) < 1 THEN VTAB 18:
PRINT "YOU HAVE NO "G$(X1)" ABOARD! ";:
GOSUB 760:
GOTO 230

300 VTAB 18:
PRINT A$;;
VTAB 18:
PRINT T$;" HOW MUCH ";G$(X1);:
PRINT "? ";:
NUM$ = "":
NUM = 0:
GOSUB 310:
GOTO 320

305 REM SOOPER DOOPER NUMBER SCOPER SUBROUTINE
(310-314)

310 PRINT CHR$(8);:
INVERSE:
PRINT " ";
NORMAL:
GOSUB 60:
IF LEN (NUM$) > 0 AND ASC (X$) = 8 THEN PRINT X$;;
PRINT" ";
PRINT X$;X$;;
PRINT" ";
FG=1:
IF LEN (NUM$) = 1 THEN NUM$ = "":
FG = 0:
GOTO 310
```

```
311 IF FG = 1 THEN NUM$ = LEFT$ (NUM$, LEN (NUM$) - 1):
      FG = 0:
      GOTO 310

312 IF ASC (X$) = 65 OR ASC (X$) = 13 THEN RETURN

313 IF ASC (X$) < 48 OR ASC (X$) > 57 THEN 310

314 NUM$ = NUM$ + X$:
      PRINT CHR$ (8);X$;;
      INVERSE:
      PRINT " ";
      NORMAL:
      GOTO 310

320 IF X = 1 THEN IF X$ = "A" THEN PRINT CHR$ (8);:
      INVERSE:
      PRINT "ALL";:
      NUM = INT (C / GP(X1))

321 IF X = 1 AND X$ <> "A" THEN NUM = VAL (NUM$)

330 IF X = 2 THEN IF X$ = "A" THEN PRINT CHR$ (8);:
      INVERSE:
      PRINT "ALL";:
      NUM = SG(X1)

331 IF X = 2 AND X$ <> "A" THEN NUM = VAL (NUM$)

335 NORMAL
```

```
340 IF X = 1 THEN IF NUM * GP(X1) > C THEN PRINT:  
    VTAB 18:  
    PRINT "YOU CAN'T AFFORD SO MUCH! ";:  
    GOSUB 760:  
    VTAB 18:  
    PRINT A$;:  
    GOTO 300  
  
341 IF X = 1 AND NUM * GP(X1) < = C  
    THEN SG(X1) = SG(X1) + NUM:  
    SH = SH - NUM:  
    C = C - GP(X1) * NUM:  
    GOSUB 130:  
    GOTO 230  
  
350 IF NUM > SG(X1) THEN VTAB 18:  
    PRINT A$;:  
    VTAB 18:  
    PRINT "YOU DON'T HAVE THAT MUCH! ";:  
    GOSUB 760:  
    VTAB 18:  
    PRINT A$;:  
    GOTO 300  
  
351 SG(X1) = SG(X1) - NUM:  
    SH = SH + NUM:  
    C = C + (NUM * GP(X1)):  
    GOSUB 130:  
    GOTO 230  
  
755 REM NO CAN DO SUBROUTINE (760)
```

```
760 PRINT "^G":
GOSUB 780:
RETURN
```

[Note the “^G” in line 760. Remember, it is typed in by first pressing the Control key, then pressing the “G” key while the Control is still pressed.]

```
775 REM DELAY SUBROUTINE (780-783)
```

```
780 POKE - 16368,0
```

```
781 FOR I = 0 TO 250
```

```
782 IF PEEK (- 16364) > 127 THEN
```

```
POKE - 16368,0:
```

```
RETURN
```

```
783 NEXT I:
RETURN
```

Here, in general, is what the lines in this chapter do:

Lines 280 and 290 “qualify” the player for buying or selling a chosen item. The No-Can-Do subroutine at 760 (and, through it, the Delay subroutine, 780) is called by these lines if the player tries to buy (280) an item costing more per unit than the player’s total cash, or sell (290) an item the player doesn’t have. Both lines route back to the Market Option routine at 230 in these situations.

Line 300 asks the player how much of an item is to be bought or sold, and GOSUBs 310 to the Sooper Dooper Number Scooper to allow the input of the amount.

Lines 310 to 314 themselves GOSUB, to 60 (the Get String Subroutine). They either collect numbers input from the keyboard, in which case

they RETURN to 300 after "RETURN" is pressed, or else take the letter "A" (for "all") as an input, and RETURN with that. Also, these lines screen out all but legitimate characters, and put an artificial cursor on the screen.

When the Sooper Dooper Number Scooper is finished, line 300 then GOTOs line 320, which, with line 321, handles buying. These lines process the "A" into an amount to be held in variable "NUM" if the "A" key was pressed, or, if a number was input, process the string (NUM\$), holding the digits of the number, into "NUM".

Lines 330 and 331 do the same thing for selling which 320 and 321 did for buying, putting a quantity into NUM.

Lines 340 and 341 either catch a player trying to buy more than cash will allow, or complete the purchase transaction, update the screen, and take the program all the way back to the Market Menu at 230.







T A | P A N

A Historical Adventure for the Apple® Computer

Keeping Our Other Options Open

CHAPTER EIGHT

Remember the Market Menu and the Market Option routines? They allowed us to choose among Buying, Selling, Leaving and Retiring. We have seen how buying and selling work. Now we'll look at what happens when the player chooses "L" for "Leave".

We get to the Other Options routine when "IF X\$="L" THEN 360" operates in line 242. Here is the start of the Other Options routine:

```
355 REM OTHER OPTIONS (360-381)

360 GOSUB 130:
      GOSUB 1340:
      VTAB 11:
      INVERSE:
      PRINT A$:
      NORMAL:
      PRINT " T)RADE, R)ECORDS";:
```

```

IF L = 0 THEN PRINT ", L)ENDER, G)ODOWN OR
E)MBARK?":
PRINT:
PRINT A$;

361 IF L <> 0 THEN PRINT " OR E)MBARK?":
PRINT:
PRINT:
PRINT A$;;

```

Line 360 first GOSUBs 130 to get an update of the Main Display, then it uses the "Cleanup subroutine" to clear the area below the Main Display. Since we'll be using the Cleanup subroutine often, let's see it now:

```

1338 REM  CLEANUP SUBROUTINE (1340-1370)

1340 FOR I = 12 TO 18

1345 HTAB 1

1350 VTAB I:
PRINT A$;

1360 NEXT I

1365 PRINT:
VTAB 12

1370 RETURN

```

All the Cleanup subroutine does is to print long lines of spaces (A\$), from VTAB 12 to VTAB 18.

Next, lines 360 and 361 show us a menu of options.

This menu is set up to display two different versions — in Hongkong, the menu will read:

T)RADE, R)ECORDS, L)ENDER, G)ODOWN OR
E)MBARK?

In any port o' call other than Hongkong, the menu will read:

T)RADE, R)ECORDS OR E)MBARK?"

Why the different menus in different ports? That's because only in our home port of Hongkong will we find the Iron Lotus Triad moneylender, Elder Brother Wu. Only there do we maintain a godown, or warehouse.

In lines 370 to 381 we continue the Other Options routine:

```

370  GOSUB 60:
      IF X$ = "G" AND L = 0 THEN X = 1
      IF X$ = "T" THEN 120
      IF X$ = "R" THEN X = 2
      IF X$ = "L" THEN X = 3
      IF X$ = "E" THEN 730
      IF (X$ <> "G" OR L <> 0) AND X$ <> "T" AND X$ <> "R" AND X$ <> "L" AND X$ <> "E" THEN GOSUB 770:
      GOTO 370

380  IF X = 3 AND L <> 0 THEN GOSUB 770:
      GOTO 370

381  IF X <> 3 OR L = 0 THEN ON X GOSUB 390,600,470:
      GOTO 360

```

Rather than always going into complete detail about programming concepts we've covered before, the Authors will assume the reader has learned (or already understood) methods used earlier. Since the above lines are similar in concept to the Market Menu and the Market Option routines, they'll require less explanation.

Line 370 uses the Get String subroutine (60) to take the player's input. Then, if the player's location is Hongkong and the input is "G" (for Godown), the flag X is given a value of 1.

In line 371, if the player had input a "T" (Trade), the program GOTOS 120 to update the screen, give the Market Prices, and the Market Menu.

Had the player input "R" (Retire), line 372 gives X the value of 2.

Likewise in line 373, if the input is "L" (Lender) the flag X is given a value of 3.

An input of "E" (Embark) causes line 374 to GOTO 730, the Embark routine.

In line 375, any input but the "legal" ones (G, T, R, L or E) are caught, the player is given the Input Error tones, and line 370 is started again from the top.

The Other Options input processing is concluded with lines 380 and 381. First line 380 checks to give the Input Error note and restarts line 370 if the player is outside of Hongkong but has tried to see the Lender.

Next, line 381 GOSUBs 390 if X=1 (Godown), GOSUBs 600 if X=2 (Retire), or GOSUBs 470 if X=3 (Lender).

Finally, after whichever subroutine was called, line 381 GOTOS line 360 to give us the Other Options Menu again.

In the next chapter we will visit our godown, and see how we can store away trade goods.

In the meantime, here are the lines we've used in this chapter:

355 REM OTHER OPTIONS (360-381)

```
360 GOSUB 130:
      GOSUB 1340:
      VTAB 11:
      INVERSE:
      PRINT A$:
      NORMAL:
      PRINT " T)RADE, R)ECORDS";:
      IF L = 0 THEN PRINT " , L)ENDER, G)ODOWN OR
      E)MBARK?":
      PRINT:
      PRINT A$;

361 IF L <> 0 THEN PRINT " OR E)MBARK?":
      PRINT:
      PRINT:
      PRINT A$;

370 GOSUB 60:
      IF X$ = "G" AND L = 0 THEN X = 1

371 IF X$ = "T" THEN 120

372 IF X$ = "R" THEN X = 2

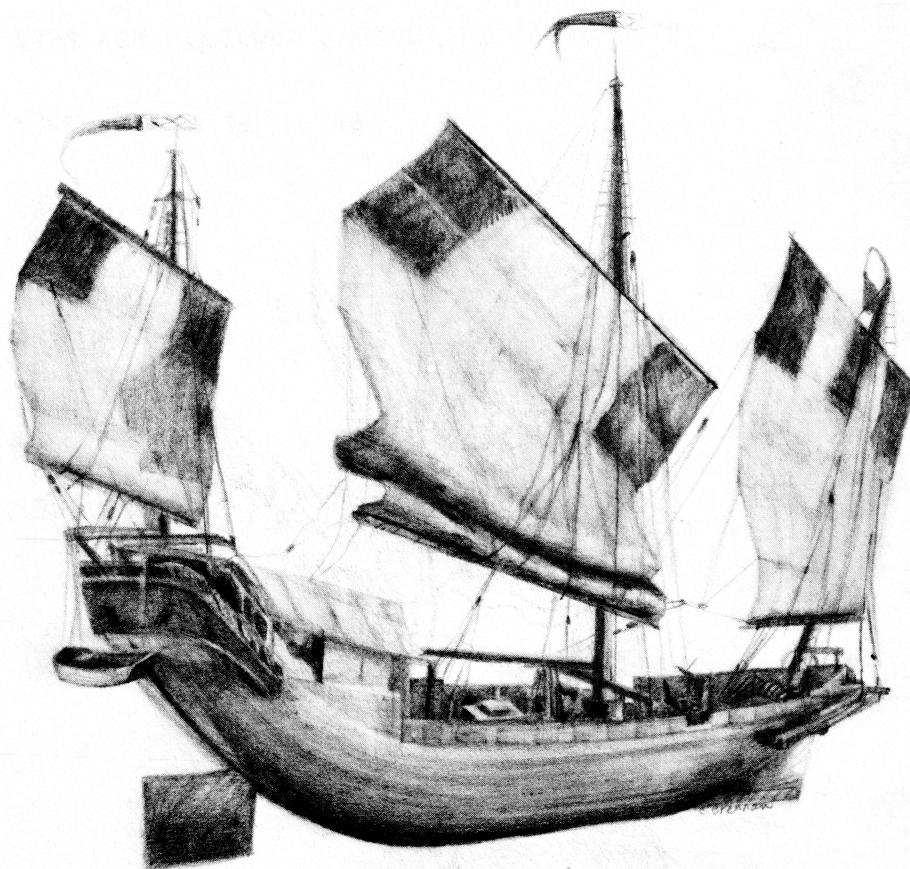
373 IF X$ = "L" THEN X = 3

374 IF X$ = "E" THEN 730

375 IF (X$ <> "G" OR L <> 0) AND X$ <> "T" AND X$ <>
      "R" AND X$ <> "L" AND X$ <> "E" THEN GOSUB 770:
      GOTO 370
```

```
380 IF X = 3 AND L <> 0 THEN GOSUB 770:  
    GOTO 370  
  
381 IF X <> 3 OR L = 0 THEN ON X GOSUB 390,600,470:  
    GOTO 360  
  
1338 REM CLEANUP SUBROUTINE (1340-1370)  
  
1340 FOR I = 12 TO 18  
  
1345 HTAB 1  
  
1350 VTAB I:  
    PRINT A$;  
  
1360 NEXT I  
  
1365 PRINT:  
    VTAB 12  
  
1370 RETURN
```





Lorchha

T A I P A N

A Historical Adventure for the Apple® Computer

Let's All Go Down to the Godown

CHAPTER NINE

The Godown subroutine is made up of eight lines, from 390 to 460. Those lines handle the transfer of trade goods to or from the godown, and trap and prevent errors by the player. Let's begin with line 390:

```
384 REM GODOWN SUBROUTINE (390-461)
```

```
387 REM GODOWN MENU (390-392)
```

```
390 GOSUB 130:  
VTAB 13:  
PRINT A$:  
VTAB 13:  
PRINT "CARGO <T>O OR <F>ROM GODOWN?":  
PRINT A$;:  
GOSUB 60:  
IF X$= "T" THEN 410
```

```
391 IF X$="F" THEN 440
```

```
392 IF X$<>"T" AND X$ <> "F" THEN GOSUB 770:
GOTO 390
```

Here we've been able to combine both a Godown Menu and the necessary processing of the player's input, into one line. If we choose to move items To the godown, the program GOTOs 410. If we want to move cargo From the godown, we GOTO 440. Any other input gives us the Input Error raspberry, and restarts line 390 to ask us again.

Lines 410 through 430 handle moving cargo to the godown. Here's line 410:

```
410 GOSUB 400:
IF SG(X1) = 0 THEN VTAB 13:
PRINT "YOU HAVE NO ";G$(X1);" ABOARD!      ":
GOSUB 760:
RETURN
```

We'll need the Godown Cargo Choice subroutine at 400, so here's that, too:

```
395 REM GODOWN CARGO CHOICE SUBROUTINE (400)
400 VTAB 13:
PRINT "MOVE WHAT ("TC$")? ":
GOSUB 260:
RETURN
```

Line 410 immediately GOSUBs 400 to get our choice of cargo to move to the godown. At line 400, we're asked which cargo item we want moved. (TC\$ is printed to prompt us with the initials of the choices: "O, S, T, A, P OR R".) The line then GOSUBs 260, where that line and 270 make up the same Item Choice subroutine we used in the Market Menu Input routine back in Chapter 6. (The policy of the Authors, in the never-ending quest for saving memory, is thus: if you can use a routine more than once, make it a subroutine.)

Line 400 uses the Item Choice subroutine to obtain a value for X1, which is a flag that stands for the chosen item. We then RETURN to line 410.

Now line 410 checks to see if we have any of the chosen item aboard ship. If not, an error message (telling us we don't have that item aboard) is displayed, the No-Can-Do subroutine has its say, and we are RETURNed to the Other Options routine.

If we do have at least one unit of the chosen item aboard our ship, line 420 asks us how much of it we'd like to move to the godown:

```

420  NUM$ = "":  

     VTAB 13:  

     PRINT A$:  

     VTAB 13:  

     PRINT " MOVE HOW MUCH ";G$(X1);"? ";:  

     GOSUB 310:  

     IF X$ = "A" THEN INVERSE:  

     PRINT CHR$(8);"ALL";:  

     NUM = SG(X1):  

     NORMAL  

421  IF X$ <> "A" THEN NUM = VAL (NUM$)  

425  HTAB 1

```

In this line one oddity needs explanation: CHR\$(8) is used to backspace over our "artificial cursor" just before printing the word "ALL".

After asking us how much to move to the godown, line 420 GOSUBs 310, where the Sooper Dooper Number Scooper subroutine gets either a number (stored as a string in NUM\$) or the letter "A", which tells the computer we want to put all of our cargo of the chosen item — SG(X1) — into the godown. Either way, we start line 430 with the numerical variable NUM holding the amount of cargo we wish to transfer from the ship to the godown:

```

430 IF NUM > SG(X1) THEN VTAB 13:
PRINT " ONLY ";
Q = SG(X1):
GOSUB 1330:
HTAB 11:
PRINT " UNITS ABOARD!      ";
GOSUB 760:
GOTO 420

431 IF NUM < = SG(X1) THEN SG(X1) = SG(X1) - NUM:
SH = SH + NUM:
GG(X1) = GG(X1) + NUM:
NUM$ = " ":
RETURN

```

First, line 430 checks to see if we're trying to move more cargo to the godown than we have aboard ship. If that's what the player is trying to do, then, after an error message and a No-Can-Do, the player is asked again, at 420, how much of the item is to be put into the godown.

If the amount to be moved is legitimate, line 430 goes on to complete the transaction, subtracting the chosen amount, NUM, from the amount aboard ship, SG(X1), and adding the same amount to the quantity of that item in the godown, GG(X1). The available cargo capacity of the ship (SH) is also increased by the number of units moved to the godown.

Finally, line 430 RETURNS us to the Other Options routine.

Moving cargo From the godown requires similar programming:

```

440 GOSUB 400:
IF GG(X1) = 0 THEN VTAB 13:
PRINT " THE GODOWN HAS NO ";G$(X1);".
GOSUB 760:
RETURN

```

```

450  NUM$ = "":  

      VTAB 13:  

      PRINT B$:::  

      HTAB 1:  

      PRINT " MOVE HOW MUCH ";G$(X1); " TO SHIP? ";:  

      GOSUB 310:  

      IF X$ = "A" THEN PRINT CHR$(8)::  

      INVERSE:  

      PRINT "ALL";:  

      NORMAL:  

      NUM = GG(X1)

451  IF X$ <> "A" THEN NUM = VAL (NUM$)

460  IF NUM > GG(X1) THEN VTAB 13:  

      PRINT " JUST ";Q = GG(X1);:  

      GOSUB 1330:  

      PRINT " UNITS STORED!":  

      GOSUB 760:  

      GOTO 450

461  IF NUM <= GG(X1) THEN GG(X1) = GG(X1) - NUM:  

      SG(X1) = SG(X1) + NUM:  

      SH = SH - NUM:  

      NUM$ = "":  

      RETURN

```

Line 440, using the subroutine at 400 (and, via 400, the subroutine at 260), asks the player which item to move from the godown to the ship. It then screens the input to make certain we have some of the item in our godown.

The amount to be moved aboard ship is determined by line 450, which calls the Sooper Dooper Number Scooper subroutine at 310, and uses either a number or the letter A for "All".

Finally, line 460 either will catch the player trying to move more of an item than is available from the godown to the ship, or will complete the transfer, updating the amounts of the item in both the ship and the warehouse, and subtracting appropriately from the ship's available capacity. It then RETURNS to the Other Options routine.

These are the lines we've input in this chapter:

```
384 REM GODOWN SUBROUTINE (390-461)

387 REM GODOWN MENU (390-392)

390 GOSUB 130:
    VTAB 13:
    PRINT A$:
    VTAB 13:
    PRINT "CARGO <T>0 OR <F>ROM GODOWN?":
    PRINT A$;;
    GOSUB 60:
    IF X$= "T" THEN 410

391 IF X$="F" THEN 440

392 IF X$<> "T" AND X$ <> "F" THEN GOSUB 770:
    GOTO 390

395 REM GODOWN CARGO CHOICE SUBROUTINE (400)

400 VTAB 13:
    PRINT "MOVE WHAT ("TC$")? ":
    GOSUB 260:
    RETURN
```

```
410 GOSUB 400:
IF SG(X1) = 0 THEN VTAB 13:
PRINT "YOU HAVE NO ";G$(X1);" ABOARD!":
GOSUB 760:
RETURN

420 NUM$ = "":
VTAB 13:
PRINT A$:
VTAB 13:
PRINT " MOVE HOW MUCH ";G$(X1);"? ";
GOSUB 310:
IF X$ = "A" THEN INVERSE:
PRINT CHR$(8);"ALL";:
NUM = SG(X1):
NORMAL

421 IF X$ <> "A" THEN NUM = VAL (NUM$)

425 HTAB 1

430 IF NUM > SG(X1) THEN VTAB 13:
PRINT " ONLY ";
Q = SG(X1):
GOSUB 1330:
HTAB 11:
PRINT " UNITS ABOARD! ";
GOSUB 760:
GOTO 420
```

```
431 IF NUM < = SG(X1) THEN SG(X1) = SG(X1) - NUM:
    SH = SH + NUM:
    GG(X1) = GG(X1) + NUM:
    NUM$ = "":
    RETURN

440 GOSUB 400:
    IF GG(X1) = 0 THEN VTAB 13:
    PRINT " THE GODOWN HAS NO ";G$(X1);":
    GOSUB 760:
    RETURN

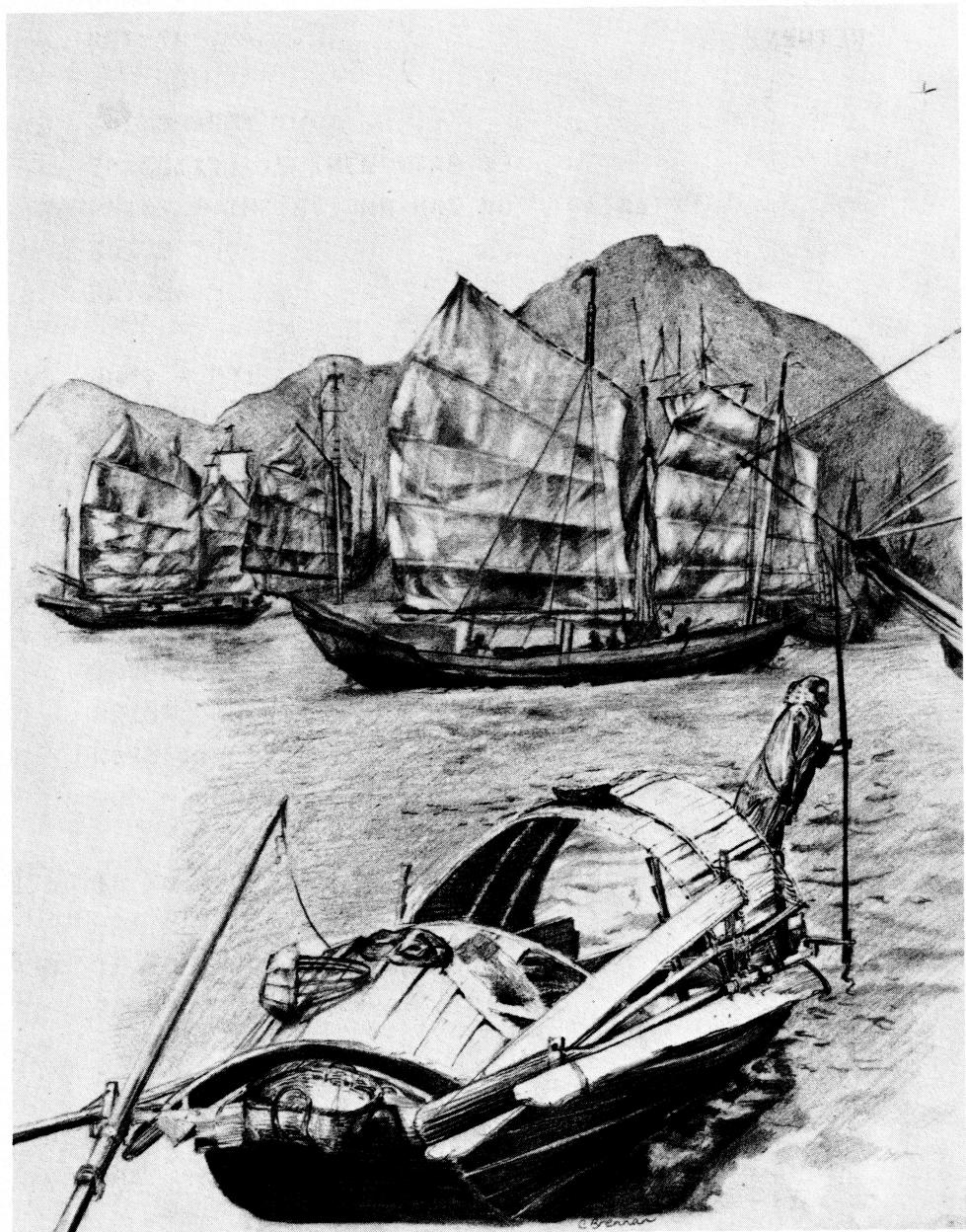
450 NUM$ = "":
    VTAB 13:
    PRINT B$;;
    HTAB 1:
    PRINT " MOVE HOW MUCH ";G$(X1);" TO SHIP? ";
    GOSUB 310:
    IF X$ = "A" THEN PRINT CHR$(8)::
    INVERSE:
    PRINT "ALL"::
    NORMAL:
    NUM = GG(X1)

451 IF X$ <> "A" THEN NUM = VAL (NUM$)

460 IF NUM > GG(X1) THEN VTAB 13:
    PRINT " JUST ";Q = GG(X1):
    GOSUB 1330:
    PRINT " UNITS STORED!":
    GOSUB 760:
    GOTO 450
```

```
461 IF NUM < = GG(X1) THEN GG(X1) = GG(X1) - NUM:  
SG(X1) = SG(X1) + NUM:  
SH = SH - NUM:  
NUM$ = ":"  
RETURN
```





Hong Kong Harbor

T A I P A N

A Historical Adventure for the Apple® Computer

An International Record Collection

CHAPTER TEN

If the player inputs "R", for Records, at the Other Options routine, the program GOSUBs to the Records subroutine.

The Records subroutine is used to display the highest and the lowest prices the player has encountered in the various ports, as well as the number of times the player has visited the ports in question. That kind of information is the most important kind of data to a trader. It gives a taipan the knowledge to plan trading routes and strategy. A player need not keep extensive records of prices. The computer will do that automatically.

Since there is nowhere near enough room on a single screen to display *all* of the highest and lowest prices the player has encountered in *all* of the ports visited, the Records subroutine gives us two modes of calling up those prices: by *Port* and by *Item*.

When using the Port mode, we are first asked for our choice of port, then we are told how many times we've been in that port, and are then given the highest and lowest prices we've yet encountered for *every time* in that port.

In the Item mode, we are first asked which item we want to know about, then we are given the number of visits we've made to each port,

and the highest and lowest prices we've encountered for the chosen item in *every port* we've visited.

In either mode, prices for items in ports we haven't visited will not be displayed, and a question mark ("?") will be shown instead. If we've been in a port only once, the high and low prices for items there will be the same. (We needn't have actually bought or sold any of the items in the records — the records for all items will be updated every time we enter a port.)

The Records subroutine starts with these lines:

```

595 REM RECORDS SUBROUTINE (600-721)

600 HOME:
    INVERSE:
    PRINT A$;;
    NORMAL:
    PRINT "      RECORDS OF HIGH & LOW PRICES":
    INVERSE:
    PRINT A$;;
    NORMAL:
    PRINT:
    PRINT "      RECORDS BY PORT OR ITEM?":
    PRINT "      _____"
    VTAB 10:
    PRINT "      PRESS <SPACEBAR> WHEN FINISHED":
    PRINT:
    INVERSE:
    PRINT A$:
    NORMAL

610 GOSUB 60:
    IF X$ = "P" THEN GOSUB 720:
    GOTO 620

```

```

611 IF X$ = "I" THEN PRINT:
  VTAB 5:
  PRINT A$;;
  PRINT A$:
  GOSUB 660:
  GOTO 670

612 IF X$ = " " THEN HOME:
  RETURN

613 IF X$ <> "P" AND X$ <> "I"
  AND X$ <> " " THEN GOSUB 770:
  GOTO 610

```

Lines 600 and 601 simply gives us a Records Menu. Note that the expected input is limited to pressing either the P, I or the "spacebar" keys.

Handling this first input are lines 610 to 612. Line 610 GOSUBs 60 to get a character to be held in X\$, then, if the input is a P, we GOSUB to the Port Choice subroutine, which will print out a list of ports from which we may choose.

The Port Choice subroutine begins at line 720:

```

715 REM PORT CHOICE SUBROUTINE (720)

720 FOR I = 0 TO 9 STEP 2:
  VTAB (I / 2) + 4:
  PRINT A$:
  VTAB (I / 2) + 4:
  PRINT I;" ";L$(I);:
  HTAB 20:
  PRINT I + 1;" ";L$I + 1):
  NEXT I:
  PRINT

```

```

721 PRINT " WHICH PORT (0-9)? ":
PRINT:
INVERSE:
PRINT A$:
NORMAL:
RETURN

```

This subroutine is very simple — it just prints out the number and the name of each port, prompts the player to input the number of the chosen port, and RETURNS to 610.

Then line 610 (still assuming that the player has chosen the "Port" mode) GOTOS 620:

```

620 GOSUB 60:
IF ASC (X$) > 47 AND ASC (X$) < 58 THEN X = VAL
(X$)

621 IF ASC (X$) < 48 OR ASC (X$) > 57 THEN GOSUB 770:
GOTO 620

```

Line 620 uses the Get String subroutine (60) to put a character into X\$, and checks that it's a numerical character. If the key pressed isn't a number, line 621 gives us the Input Error note, and then we are given another chance. Otherwise, the numerical character in X\$ is converted by line 620 into a value in X, and we continue to lines 630, 640, 641 and 650:

```

630 HOME:
PRINT " ";L$(X);" PRICES (VISITS: ";V(X);") ":
INVERSE:
PRINT " ITEM HIGH LOW ":
NORMAL:
FOR I = 0 TO 5:
PRINT " ";G$(I);:
HTAB 21

```

```

640 IF H(X,I) = 0 THEN PRINT " ?";:
  HTAB 30:
  INVERSE:
  PRINT " ";
  NORMAL:
  PRINT " ?";
  NEXT I

641 IF H(X,I) <> 0 THEN HTAB 21:
  PRINT H(X,I);:
  HTAB 30:
  INVERSE:
  PRINT " ";
  NORMAL:
  PRINT " ";L(X,I):
  NEXT I

650 INVERSE:
  PRINT A$:
  NORMAL:
  GOTO 700

```

Here's where the list of high and low prices for each item in the selected port is displayed. This routine could be much simpler if we'd not bothered to put in graphics to format it neatly, and if we'd been willing to just show the prices as "0" for any port the player hadn't visited. But the Authors are convinced that graphics can enhance the readability of a screen, and that putting in question marks for prices in ports we haven't visited is more accurate and much less misleading.

In line 630 we first HOME to clear the screen. Then we display at the top of the screen, the name of the port, L\$(X), the word "PRICES", and then the number of times we've visited the chosen port. (Remember that the numerical variable, X, now holds the number of the port we've chosen.) This serves to give us a heading for the information which will follow. That heading might look something like this:

SINGAPORE PRICES (VISITS: 2)

Line 630 then continues with a line using INVERSE characters embedded in a bright bar. This line serves to identify the three vertical columns which will go below. Thus far, we might have something like the following displayed on the screen:

```
SINGAPORE PRICES (VISITS: 2)
#ITEM#####HIGH#####LOW#####
```

Finally, line 630 starts a loop which increments I from 0 to 5, and starts printing the names of our six trade goods.

The start of line 640 checks to see if the highest price for any item in the chosen port is zero. This can be true only if the variable which holds the high price, H(X,I), has never been given a value since initialization. And *that* can only occur if we've never been to this port. So if the value of H(X,I) is zero, line 640 prints question marks (with bright blocks between them) instead of the high and low prices for such items. (In the Port mode, the Records subroutine will naturally show *only* question marks for *all* items in a port we haven't visited.)

If the high and low prices are known, line 641 goes on to display those prices.

A bright bar (an INVERSE A\$) is displayed by line 650, and we then GOTO line 700.

Lines 700 to 711 are used simply to allow us to look at the display of prices, then press the spacebar when we've seen enough:

```
700 PRINT " PRESS <SPACEBAR> WHEN FINISHED";
710 GOSUB 60:
  IF X$ = " " THEN 600
711 GOTO 710
```

The total screen in the Port mode of the Records subroutine should look like this to us:

```

SINGAPORE PRICES (VISITS: 2)
#ITEM#####HIGH#####LOW#####
OPIUM           26213  # 16504
SILK            10287  # 7642
TEA              1362   # 1199
ARMS            231    # 226
PEPPER          29     # 27
RICE             ?      # 6
#####
PRESS <SPACEBAR> WHEN FINISHED

```

When the spacebar is pressed, line 710 throws us right back to line 600, where we once again have the choice of viewing high and low prices either in the Port or Item mode. This time, let's suppose we choose the Item mode. Line 611 will first have us GOSUB 660:

```

660  VTAB 5:
FOR I = 0 TO 5:
PRINT " ";G$(I),:
NEXT I:
PRINT:
PRINT " WHAT ITEM (";TC$; "?":
PRINT:
PRINT A$:
INVERSE:
PRINT A$:
NORMAL:
GOSUB 260:
RETURN

```

This line displays the names of all the items of trade, and asks the player for which item price records are to be displayed. It then GOSUBs 260 (the same Item Choice subroutine used by both the Market and the Godown routines), and thus RETURNS briefly to 611 with the chosen item's number held in variable X1.

Next, line 611 immediately GOTOS line 670, where the Item mode display starts, continuing with lines 680, 681 and 690:

```

670: HOME:
      PRINT " ";G$(X1);" PRICES":
      INVERSE:
      PRINT " PORT      VISITS      HIGH      LOW      ";
      NORMAL:
      FOR I = 0 TO 9:
      PRINT L$(I);

680  VTAB I + 3:
      HTAB 12:
      PRINT V(I);:
      IF L(I,X1) = 0 THEN HTAB 21:
      INVERSE:
      PRINT " ";
      NORMAL:
      PRINT " ?";
      HTAB 32:
      PRINT "?";
      NEXT I:
      GOTO 690

681  IF L(I,X1) <> 0 THEN HTAB 21:
      INVERSE:
      PRINT " ";
      NORMAL:
      PRINT " ";H(I,X1);:
      HTAB 31:
      PRINT " ";L(I,X1);:
      NEXT I

690  INVERSE:

```

PRINT A\$:

NORMAL

These lines are so similar to the routine used in the Port mode that they require no further explanation. The program "falls through" to lines 700, 710 and 711, which we used in the Port mode to terminate the display. Here's how an Item mode display might look:

TEA PRICES

#PORT#####VISITS#####HIGH#####LOW#####

HONGKONG	2	#	705	667
FOOCHOW	0	#	?	?
SHANGHAI	0	#	?	?
NAGASAKI	0	#	?	?
MANILA	0	#	?	?
SINGAPORE	3	#	1362	1199
BATAVIA	1	#	1368	1368
SAIGON	0	#	?	?
CALCUTTA	1	#	1777	1777
LIVERPOOL	0	#	?	?

######

PRESS <SPACEBAR> WHEN FINISHED

We can now hit the spacebar once, and we're back to the Records Menu, and hit it again, and we RETURN all the way back to 381, which in turn has us immediately GOTO 360, the Other Options routine. Now we're ready to explore some of the other Other Options. But first, here are all the lines we've put into our computer in this chapter:

595 REM RECORDS SUBROUTINE (600-721)

600 HOME:

INVERSE:

PRINT A\$::

NORMAL:
PRINT " RECORDS OF HIGH & LOW PRICES":
INVERSE:
PRINT A\$;:
NORMAL:
PRINT:
PRINT " RECORDS BY PORT OR ITEM?":
PRINT " _____"

601 VTAB 10:
PRINT " PRESS <SPACEBAR> WHEN FINISHED":
PRINT:
INVERSE:
PRINT A\$:
NORMAL

610 GOSUB 60:
IF X\$ = "P" THEN GOSUB 720:
GOTO 620

611 IF X\$ = "I" THEN PRINT:
VTAB 5:
PRINT A\$;:
PRINT A\$:
GOSUB 660:
GOTO 670

612 IF X\$ = " " THEN HOME:
RETURN

613 IF X\$ <> "P" AND X\$ <> "I"
AND X\$ <> " " THEN GOSUB 770:
GOTO 610

```
620 GOSUB 60:
  IF ASC (X$) > 47 AND ASC (X$) < 58 THEN X = VAL
  (X$)

621 IF ASC (X$) < 48 OR ASC (X$) > 57 THEN GOSUB 770:
  GOTO 620

630 HOME:
  PRINT " ";L$(X);" PRICES (VISITS:";V(X);")":
  INVERSE:
  PRINT " ITEM           HIGH           LOW           ";
  NORMAL:
  FOR I = 0 TO 5:
  PRINT " ";G$(I);:
  HTAB 21

640 IF H(X,I) = 0 THEN PRINT " ?":
  HTAB 30:
  INVERSE:
  PRINT " ";
  NORMAL:
  PRINT " ?":
  NEXT I

641 IF H(X,I) <> 0 THEN HTAB 21:
  PRINT H(X,I);:
  HTAB 30:
  INVERSE:
  PRINT " ";
  NORMAL:
  PRINT " ";L(X,I);:
  NEXT I
```

```
650 INVERSE:
PRINT A$:
NORMAL:
GOTO 700

660 VTAB 5:
FOR I = 0 TO 5:
PRINT " ";G$(I);:
NEXT I:
PRINT:
PRINT " WHAT ITEM (";TC$; "?":
PRINT:
PRINT A$:
INVERSE:
PRINT A$:
NORMAL:
GOSUB 260:
RETURN

670 HOME:
PRINT " ";G$(X1);" PRICES":
INVERSE:
PRINT " PORT      VISITS      HIGH      LOW      ";;
NORMAL:
FOR I = 0 TO 9:
PRINT L$(I);

680 VTAB I + 3:
HTAB 12:
PRINT V(I);:
IF L(I,X1) = 0 THEN HTAB 21:
INVERSE:
```

```
PRINT " ";
NORMAL:
PRINT " ?";
HTAB 32;
PRINT "?";
NEXT I;
GOTO 690

681 IF L(I,X1) <> 0 THEN HTAB 21;
INVERSE:
PRINT " ";
NORMAL:
PRINT " ";H(I,X1);;
HTAB 31;
PRINT " ";L(I,X1);
NEXT I

690 INVERSE:
PRINT A$;
NORMAL

700 PRINT " PRESS <SPACEBAR> WHEN FINISHED";

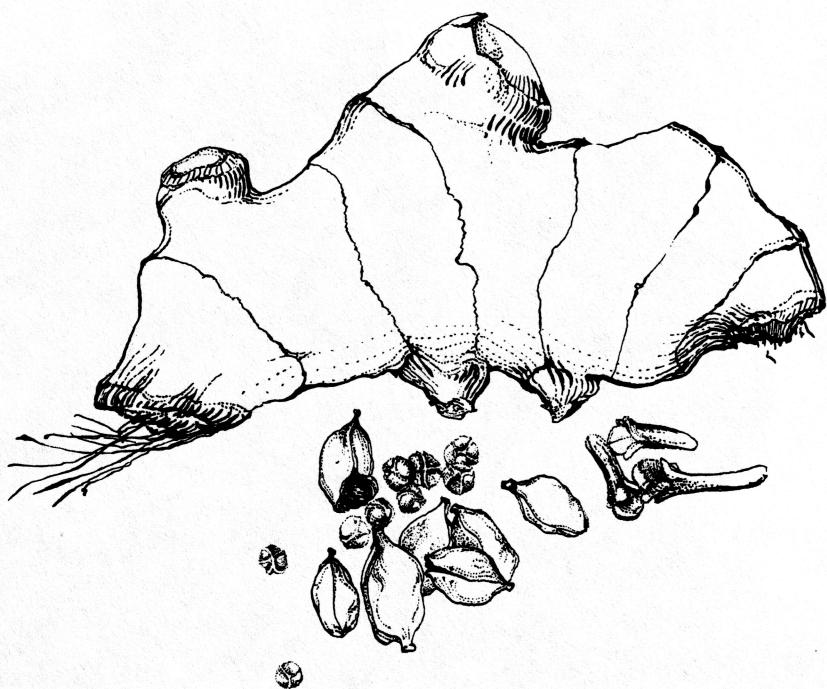
710 GOSUB 60;
IF X$ = " " THEN 600

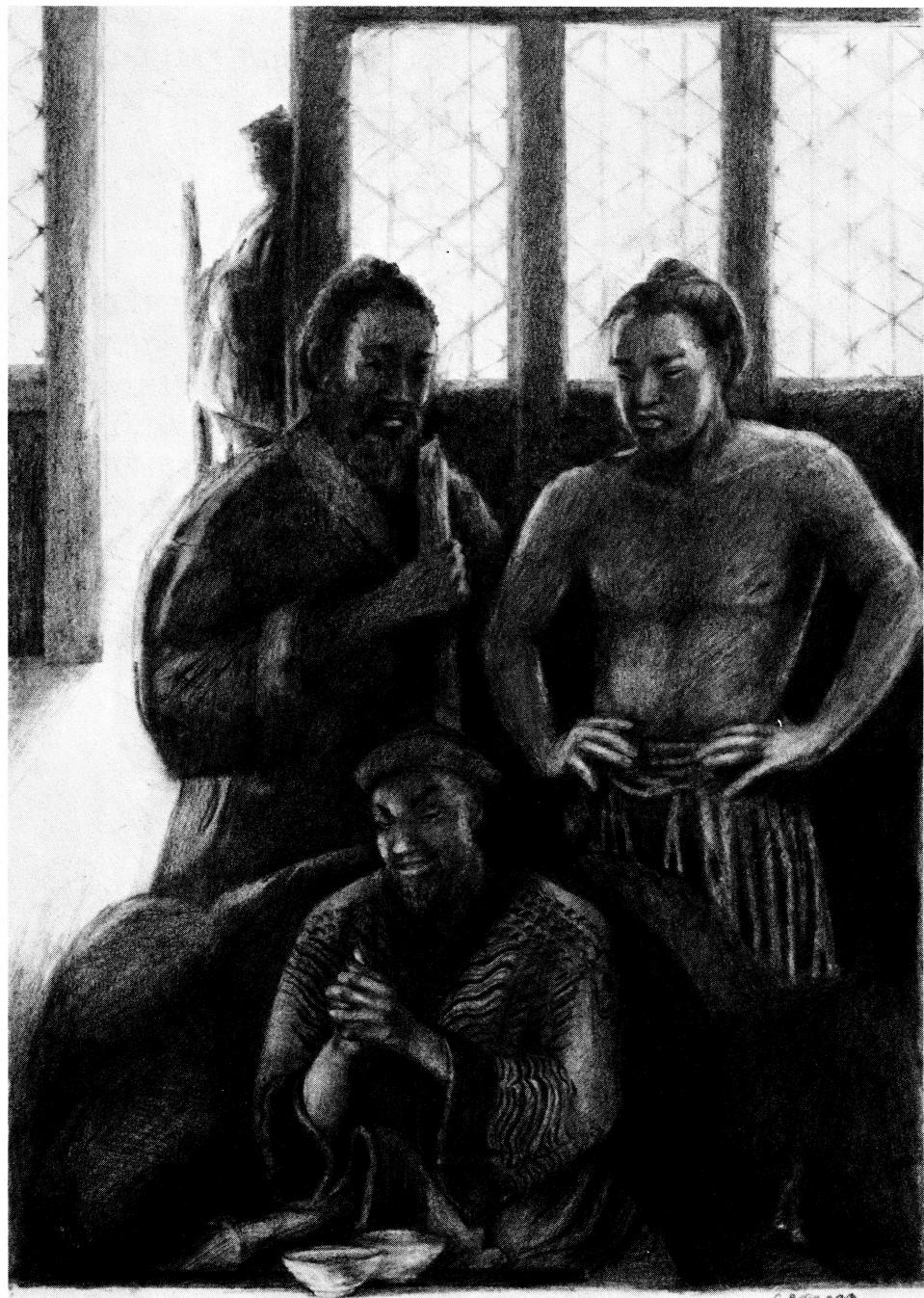
711 GOTO 710

715 REM PORT CHOICE SUBROUTINE (720)

720 FOR I = 0 TO 9 STEP 2;
VTAB (I / 2) + 4;
```

```
PRINT A$:  
VTAB (I / 2) + 4:  
PRINT I;" ";L$(I);:  
HTAB 20:  
PRINT I + 1;" ";L$I + 1):  
NEXT I:  
PRINT  
  
?21 PRINT " WHICH PORT (0-9)?  
PRINT:  
INVERSE:  
PRINT A$:  
NORMAL:  
RETURN
```





Elder Brother Wu at the office

T A I P A N

A Historical Adventure for the Apple® Computer

Brother, Can You Spare a Dime?

CHAPTER ELEVEN

Choosing "L" for lender at the Other Options routine is, like the Godown option, possible only when we are in Hongkong. It brings us face to face with the chief of the Iron Lotus Triad, Elder Wu.

The Iron Lotus Triad has come a long way under the leadership of our fictitious Elder Brother Wu. In the "old days", when the triad had been headquartered in Foochow, spies sent by the hated Manchus had constantly acted as agents provocateur, sparking poorly-planned riots and rebellions. As a result, members and leaders of the Iron Lotus had often been caught, tortured, and beheaded. The Triad's treasury was regularly "impounded" by thieving officials who kept it all for themselves.

But Elder Brother Wu had changed all that. He had brought the headquarters of the Iron Lotus Triad to Hongkong. Secure Hongkong, where the British barbarians kept the Manchu barbarians at a safe distance. Here Wu had established secret schools which taught his braves the deadly Iron Lotus variant of Chinese boxing. A base from which the Iron Lotus could spread its underworld and political tentacles far into China and the Chinese communities of Southeast Asia. A prosperous haven, where Wu had transformed the financial aid services the triad provided

to members into a lucrative moneylending business, which bankrolled even some of the Western barbarian traders.

When we press the "L" key on our Apple II, we must remember it means we're visiting *this* Brother Wu.

Line 381 of the Other Options routine sends us to the beginning of the Lender routine:

```
465 REM LENDER SUBROUTINE (470-580)

470 GOSUB 130:
      GOSUB 1340:
      VTAB 12:
      PRINT "WANCHAI DISTRICT OF HONGKONG: HOME OF    ";
      W$;" ]":
      PRINT W$;" GREETS YOU, TAIPAN, AND WISHES YOU
      WELL.":
      GOSUB 780:
      VTAB 14:
      PRINT A$:
      PRINT A$
```

This line amounts to a simple greeting by Wu, followed by the use of the Delay subroutine to give the player time to read it — or the option of cutting the message short using the spacebar.

Way back in line 10, we'd put the words, "ELDER BROTHER WU" into W\$. This string will be used so often in the Lender routine that the memory saved by this technique should become obvious.

Wu's greeting continues in line 480:

```
480 VTAB 14:
      PRINT "WU STATES THAT HIS IRON LOTUS":
      PRINT "TRIAD HAS BEEN WATCHING YOU.":
      GOSUB 780
```

We finally cut the social graces and get down to business in the next five lines:

```

490  VTAB 14:
      PRINT A$:
      PRINT A$:
      PRINT A$:
      VTAB 14:
      PRINT W$;" ASKS, DO YOU":
      PRINT A$:
      VTAB 15:
      PRINT "WISH TO BORROW, PAY, OR QUIT? ":
      NUM$ = ""

```

```

500  GOSUB 60:
      IF X$ = "B" THEN LD$ = "BORROW":
      LD = 1

```

```

501  IF X$ = "P" THEN LD$ = "PAY":
      LD = 2

```

```

502  IF X$ = "Q" THEN PRINT:
      GOSUB 1340:
      RETURN

```

```

503  IF X$ <> "B" AND X$ <> "P"
      AND X$ <> "Q" THEN GOSUB 770:
      GOTO 500

```

These lines are typical of the kind of menu and menu-handling routines we're using in Taipan. Note the use of LD\$, which will allow the dual use of a later line (to save memory, as usual). We're also setting the variable LD as a flag to indicate what type of business we're doing with

Wu. The input of "Q" is our escape route from the home of Elder Brother Wu. It puts us back at the Other Options routine.

If we've decided to borrow money from Wu, line 510 represents that gentleman's decisionmaking as to whether or not he wants to loan us *anything* at the moment:

```
510 IF LD = 1 AND (B = 1 OR D > 1E4) THEN GOSUB 1340:
    VTAB 15:
    PRINT "WU REGRETS THAT HE CANNOT LOAN":
    PRINT "YOU MORE AT THIS TIME, TAIPAN.";;
    GOSUB 760
    GOTO 490
```

Elder Brother Wu's logic here translates as: "That barbarian wants to borrow money from me, eh? Then I'll check to see if he has *either* borrowed previously this time in port, *or* already owes me more than 10,000 in cash. If so, I'll tell him to fly a kite — but politely."

The "B=1" in line 510 is a check of a flag which is set later in the Lender routine to indicate that the player has borrowed from Wu. (B is cleared to a value of 0 whenever we enter port.)

If we get past this initial rejection, the next three lines will process the input of the amount to be borrowed or repaid:

```
520 GOSUB 1340:
    VTAB 15:
    PRINT "HOW MUCH DO YOU WISH TO ";LD$;"?":
    PRINT "TAIPAN? ";;
    GOSUB 310:
    NUM = VAL (NUM$)

530 IF X$ = "A" AND LD = 1 THEN PRINT CHR$ (8);:
    INVERSE:
    PRINT "ALL";:
    NORMAL:
    NUM = C * 2
```

```
531 IF X$ = "A" AND LD = 2 THEN PRINT CHR$(8)::  
INVERSE:  
PRINT "ALL";:  
NORMAL:  
NUM = D
```

Line 520 asks the quantity of cash we wish to either borrow from or pay to Wu. (Lines 500 and 501 had put either "BORROW" or "PAY" into LD\$.) Next, the line GOSUBs the Sooper Dooper Number Scooper, then converts any number characters in NUM\$ into a value to be held in NUM.

Then lines 530 and 531 handle, respectively, the input of "A" for "All" for borrowing, and for paying. Line 530 sets the amount to be borrowed to *twice the player's cash (C)*. Line 531 sets the amount to be repaid to the full amount owed (D). (But what if we've got less cash than we owe? We'll handle that possibility in a moment.)

Line 540 routes program flow to line 560, if we're paying Elder Brother Wu. If we're borrowing, line 541 checks to see if we're trying to borrow more than our credit limit with Wu will allow:

```
540 IF LD = 2 THEN 560  
  
541 IF NUM > 2 * C THEN VTAB 15:  
PRINT A$:  
VTAB 15:  
PRINT W$;" REGRETS THAT HE":  
PRINT "CANNOT LOAN YOU THAT MUCH."::  
GOSUB 760:  
GOTO 490:
```

Wu's simple formula of lending is thus: you can borrow only twice what you already have in cash. You show him your cash, he gives you his.

The convenience of this credit check comes free — with his 100% interest rates.

(Notice, again, that if we're paying Wu, line 540 has the program GOTO 560, skipping the lines between.)

The B flag is set in line 550, to indicate we've borrowed once this time in Hongkong. Here's the whole line:

```
550  B = 1:
      C = C + NUM:
      D = D + NUM:
      GOSUB 130:
      GOTO 490
```

Line 550 adds the borrowed money to our cash, adds the same amount to our debt, updates the screen to show these changes, then tosses us back to the main Lender menu at 490.

The rest of the Lender routine handles paying Wu. Remember the question we raised about what to do if we tried to pay back more than we had in cash? Lines 560 and 561 take care of that, and a few other things as well:

```
560  IF NUM > C THEN NUM = C:
      D = D - C:
      C = 0:
      GOSUB 130:
      VTAB 15:
      PRINT W$;" THANKS YOU.":
      PRINT "TAIPAN, FOR THE PAYMENT.":;
      GOSUB 780:
      IF D < 0 THEN D = 0:
      GOSUB 130:
      GOTO 490

561  IF NUM > C AND D > = 0 THEN GOSUB 130:
      GOTO 490
```

First, we've checked to determine whether we're trying to repay Wu more money than we have. In such a case, Wu takes all of our cash, our cash and our debt are corrected, and we are given Wu's thanks. If, additionally, we've paid Wu more than we owe him, line 560 has him consider accounts even. (Wu will not consider himself in debt to *us*!). The player, like a real taipan, will just have to be careful.) In either case, either at the end of line 560 or line 561, we're now routed back to the main Lender menu.

In line 570 the lender routine checks for our having input a payment which is *larger* than our debt, yet is *less than or equal* to our cash. (We know that the payment will be less than our cash because the program wouldn't have gotten past the last line if it weren't.)

Here is how line 570 handles its task:

```
570  IF NUM > D THEN D = 0:
      C = C - NUM:
      VTAB 15:
      PRINT W$;" THANKS YOU FOR YOUR STARTLING GENEROS
ITY!";:
      GOSUB 130:
      GOSUB 780:
      GOTO 490
```

Wu has just taken what we offered to pay him. He's grateful, because he can now apply this windfall toward the business of the Iron Lotus Triad. And we can't get the extra money back!

The more usual type of repayment is handled in line 580:

```
580  C = C - NUM:
      D = D - NUM:
      GOSUB 130:
      VTAB 15:
      PRINT W$;
```

```

" ACCEPTS YOUR PAYMENT WITH GRATITUDE, TAIPAN
";:
GOSUB 780:
GOTO 490

```

Here we've paid Wu with money we actually have, and have paid only all or part of what we owe him. The program takes us back to line 490, where we can Quit the Lender routine and go back to the Other Options routine.

These are the program lines we've introduced in this chapter:

```

465 REM LENDER SUBROUTINE (470-580)

470 GOSUB 130:
      GOSUB 1340:
      VTAB 12:
      PRINT "WANCHAI DISTRICT OF HONGKONG: HOME OF      ";
      W$;"":"
      PRINT W$;" GREETS YOU, TAIPAN, AND WISHES YOU
      WELL.":
      GOSUB 780:
      VTAB 14:
      PRINT A$:
      PRINT A$:

480 VTAB 14:
      PRINT "WU STATES THAT HIS IRON LOTUS":
      PRINT "TRIAD HAS BEEN WATCHING YOU.":
      GOSUB 780

490 VTAB 14:
      PRINT A$:
      PRINT A$:
      PRINT A$:

```

```
VTAB 14:
PRINT W$;" ASKS, DO YOU?":
PRINT A$;;
VTAB 15:
PRINT "WISH TO BORROW, PAY, OR QUIT?    ";
NUM$ = ""

500 GOSUB 60:
IF X$ = "B" THEN LD$ = "BORROW":
LD = 1

501 IF X$ = "P" THEN LD$ = "PAY":
LD = 2

502 IF X$ = "Q" THEN PRINT:
GOSUB 1340:
RETURN

503 IF X$ <> "B" AND X$ <> "P"
AND X$ <> "Q" THEN GOSUB 770:
GOTO 500

510 IF LD = 1 AND (B = 1 OR D > 1E4) THEN GOSUB 1340:
VTAB 15:
PRINT "WU REGRETS THAT HE CANNOT LOAN":
PRINT "YOU MORE AT THIS TIME, TAIPAN.";;
GOSUB 760:
GOTO 490

520 GOSUB 1340:
VTAB 15:
PRINT "HOW MUCH DO YOU WISH TO ";LD$;",":
PRINT "TAIPAN? ";
GOSUB 310:
NUM = VAL (NUM$)
```

530 IF X\$ = "A" AND LD = 1 THEN PRINT CHR\$ (8)::

INVERSE:

PRINT "ALL";:

NORMAL:

NUM = C * 2

531 IF X\$ = "A" AND LD = 2 THEN PRINT CHR\$ (8)::

INVERSE:

PRINT "ALL";:

NORMAL:

NUM = D

540 IF LD = 2 THEN 560

541 IF NUM > 2 * C THEN VTAB 15:

PRINT A\$:

VTAB 15:

PRINT W\$;" REGRETS THAT HE":

PRINT "CANNOT LOAN YOU THAT MUCH."::

GOSUB 760:

GOTO 490

550 B = 1:

C = C + NUM:

D = D + NUM:

GOSUB 130:

GOTO 490

560 IF NUM > C THEN NUM = C:

D = D - C:

C = 0:

GOSUB 130:

VTAB 15:

PRINT W\$;" THANKS YOU.":

PRINT "TAIPAN, FOR THE PAYMENT.";;

GOSUB 780:

IF D < 0 THEN D = 0:

GOSUB 130:

GOTO 490

561 IF NUM > C AND D > = 0 THEN GOSUB 130:
GOTO 490

570 IF NUM > D THEN D = 0:
C = C - NUM:
VTAB 15:
PRINT W\$;" THANKS YOU FOR YOUR STARTLING GENEROS
ITY!";;
GOSUB 130:
GOSUB 780:
GOTO 490

580 C = C - NUM:
D = D - NUM:
GOSUB 130:
VTAB 15:
PRINT W\$;
" ACCEPTS YOUR PAYMENT WITH GRATITUDE, TAIPAN
";;
GOSUB 780:
GOTO 490





© Branner

T A I P A N

A Historical Adventure for the Apple® Computer

A Gold Watch and a Hearty Handshake

CHAPTER TWELVE

The last option available to us at the Other Options menu is "E" for Embark — but that option (and all the things that can happen at sea) will require several chapters to cover effectively. So let's skip "Embark" until later. Instead, we'll press "T" for Trade, which gets us to the Market Menu.

If we were now to input "R", for Retire, the program will GOTO the That's All Folks routine.

The That's All Folks routine is the retirement and scoring portion of Taipan. The first line of the routine is 1300:

```
1295 REM THAT'S ALL FOLKS (1300-1321)
```

```
1300 HOME:
```

```
NW = C - D:
```

```
Q = NW / GT:
```

```
VTAB 4:
```

```
INVERSE:
```

```

PRINT A$;;
NORMAL:
PRINT:
PRINT "YOUR SCORE, BASED UPON TIME AND YOUR";
PRINT "NET WORTH (EXCLUDING STOCK) IS ";;
GOSUB 1330:
INVERSE:
PRINT A$:
NORMAL

```

This pretty well establishes that the object of the game of Taipan is to make as much money as possible in the least possible time. Line 1300 first clears the screen. Next, the variable NW (for Net Worth) is defined as being our cash minus our debt. Then NW is divided by GT (Game Time), which has been updated each time we've gone on an ocean voyage. The result of this division is placed into the variable Q, so it can be displayed by the Big Number subroutine. The line goes on to give its message and to display the player's score.

The following line is used to allow the player to return to playing the game if the That's All Folks routine was reached via the Market Menu, rather than by the player's "death" by pirates or shipwreck. (We'll see those less pleasant ways of getting there in later chapters.) By using line 1310, we are able to do two things — allow a player to recover from an accidental input of "R" at the Market routine, and permit the player to check out his or her score in the middle of the game.

This is how we've done it:

```

1310 IF X$ = "R" THEN PRINT "WOULD YOU LIKE TO PICK UP
THIS";
PRINT "GAME WHERE YOU LEFT OFF (Y/N)?";
GOSUB 60:
IF X$ = "Y" THEN HOME:
GOTO 120

```

The routines which might have "killed" us are designed in such a way that X\$ would never contain an "R" if we were to be wiped out at sea. So

we use the presence of an "R" in X\$ to let us know that we'd reached this routine voluntarily, from the Market Menu. If that's the case, line 1310 operates. (Otherwise the program fails through to the next line.)

If we now press the "Y" key, we are taken right back to the Market Menu, with the game continuing as if nothing had happened.

If, however, we were either killed off in the game, or had not chosen to continue the game, lines 1320 and 1321 come into play:

```
1320 GOSUB 1340
    VTAB 10:
    PRINT A$;;
    PRINT "DO YOU WISH TO START OVER (Y/N)/":
    GOSUB 60:
    IF X$ = "Y" THEN RUN
1321 END
```

All we do here is give the player a chance to start the game over from the top. This way we don't have to enter "RUN" every time we play the game.

Here are the lines we've used in this short chapter:

```
1295 REM THAT'S ALL FOLKS (1300-1321)
```

```
1300 HOME:
    NW = C - D:
    Q = NW / GT:
    VTAB 4:
    INVERSE:
    PRINT A$;;
    NORMAL:
    PRINT:
    PRINT "YOUR SCORE, BASED UPON TIME AND YOUR":
    PRINT "NET WORTH (EXCLUDING STOCK) IS ";:
    GOSUB 1330:
```

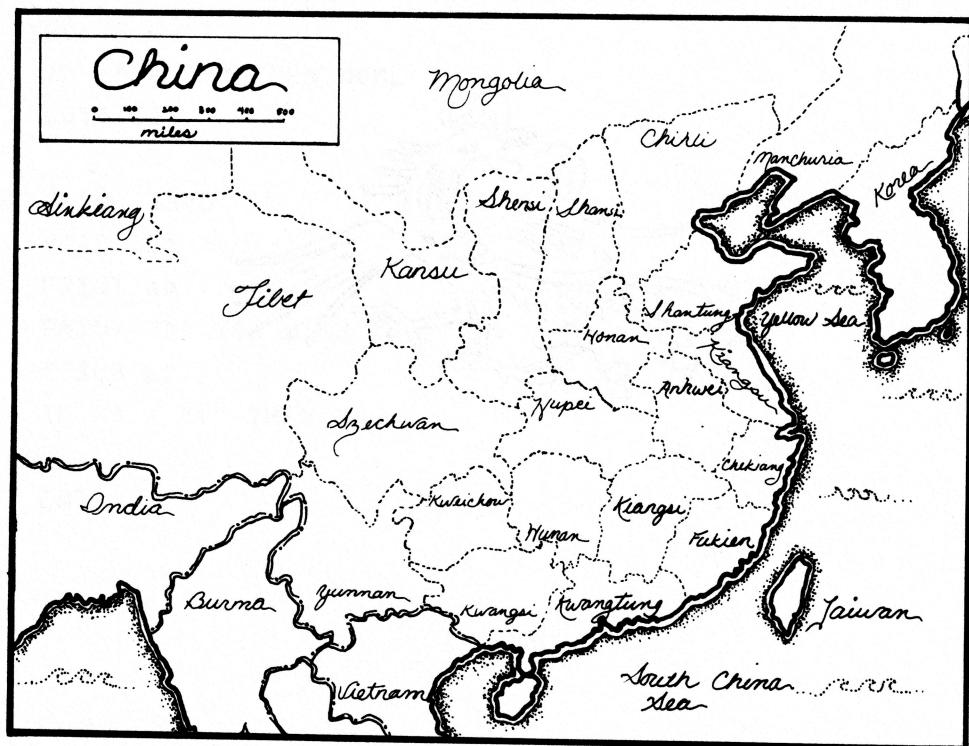
INVERSE:
PRINT A\$:
NORMAL

1310 IF X\$ = "R" THEN PRINT "WOULD YOU LIKE TO PICK UP
THIS":
PRINT "GAME WHERE YOU LEFT OFF (Y/N)?":
GOSUB 60:
IF X\$ = "Y" THEN HOME:
GOTO 120

1320 GOSUB 1340
VTAB 10:
PRINT A\$:::
PRINT "DO YOU WISH TO START OVER (Y/N)/*":
GOSUB 60:
IF X\$ = "Y" THEN RUN

1321 END





T A I P A N

A Historical Adventure for the Apple® Computer

Anchors Aweigh!

CHAPTER THIRTEEN

Now let's go back to the one choice in the Other Options routine which we haven't yet explored: "E" for "Embark".

We previously took a geographical history tour of the ten ports o' call in Taipan. Now it's time for the player to visit some of them. The Other Options routine has us GOTO 730 to begin our voyage:

```
725 REM EMBARK (730-751)  
  
730 IF SH < 0 THEN VTAB 13:  
    PRINT "YOUR SHIP IS OVERLOADED, TAIPAN      ";  
    PRINT A$:  
    GOSUB 760:  
    GOTO 360
```

```
731 IF SH > = 0 THEN HOME:  
PRINT TAB( 11)::  
INVERSE:  
PRINT "EMBARKING":  
NORMAL:  
PRINT TAB( 9);"FROM ";L$(L):  
INVERSE:  
PRINT A$:  
NORMAL:  
FOR I = 0 TO 9:  
IF L = I THEN NEXT I:  
GOTO 740  
  
732 IF L <> I THEN PRINT TAB( 10);I;" ";L$(I):  
NEXT I
```

Line 730 begins by preventing us from sailing our ship with more than a capacity load of cargo. The variable SH holds the *available* capacity of the craft, not the load the ship actually has aboard. In other words, if our ship has a capacity of 50 units, a cargo of 23 units of rice and 20 units of arms would give us an available capacity of 7 units. If we were to add 8 units of pepper, we could not sail. With an overload, we get a message display, a No-Can-Do buzz, and the program will GOTO 360, the Other Options routine. We must either transfer excess cargo to the godown (if we're in Hongkong), or sell enough to allow us to sail.

Think of excess cargo as not actually being aboard ship, but standing in stacks of cases on the dock. Thus we can have any amount of cargo temporarily "aboard ship" while in port. But we're not able to leave it on the docks when we embark. The cargo must either be in our godown, or must fit aboard ship. Eventually, if the player prospers as a taipan, it will be possible to obtain a larger ship. But cargo capacity will be a limiting bottleneck at many points throughout the game—a bottleneck which is necessary to ensure challenging play.

But if we're not overloaded, lines 731 and 732 go on to give us a display which shows us the port from which we're embarking, and then run through a loop which prints the numbers and names of all the *other* ports.

Line 740 prompts the player to select the next port o' call:

```
740 PRINT:
PRINT:
INVERSE:
PRINT A$:
NORMAL:
PRINT "WHAT PORT O' CALL, TAIPAN (0-9)?"
```

After the prompt, the next line processes the player's choice:

```
750 GOSUB 60:
IF ASC (X$) > 47 AND ASC (X$) < 58 AND VAL (X$) <>
L
THEN PO = VAL (X$):
GOTO 980

751 GOSUB 770:
GOTO 750
```

Line 750 makes sure that the input character is a number, and that it is *not* the number of the port the player is in. If the input is not a number, or is the number of the port we're in (L), line 751 has us GOSUB the Input Error subroutine, and then try for another input. But if the input is acceptable, the program will put the number of the chosen port into variable PO, then it will GOTO 980, where the Voyage routine begins:

```
975 REM VOYAGE (980-1290)
```

```

980 HOME:
PRINT:
INVERSE:
PRINT A$;;
NORMAL:
PRINT " SEA VOYAGE FROM ";L$(L):
PRINT " TO ";L$(PO):
INVERSE:
PRINT A$:
NORMAL:
GOSUB 780:
HOME:
ET = ABS (LO(L) - LO(PO))

```

```

981 IF INT ( RND (1) * 50 ) + 1 >
    INT ( RND (1) * ET ) + 1 THEN 1290

```

These lines do several things. First, line 980 shows us where we've been and where we're going. Last, it computes the raw value for ET, the "Elapsed Time" (in days) of the voyage. This uses a tricky method based upon subtracting the "location" figure of the port we're leaving, "LO(L)" from the location number of the next port o' call, "L(PO)". The resulting number is then made an "absolute number"—any minus sign is removed. That gives us the raw number of days required for the voyage. (Later, that figure will be processed further to give a bit of randomness to the voyage's duration.)

Line 981 then makes a decision as to whether or not there might be "trouble" during the voyage, by comparing the value of a random fraction times fifty (plus one), to the value of another random fraction multiplied by "ET" (plus one). Here we use a technique which the Authors favor: a combination of controllable and random factors. Since the length of the voyage can be controlled by the choice of port o' call, there is an element we might loosely call "free will". And since a random factor operates on a number we have "chosen", there is sort of a "fate" factor as well. The longer the voyage, the *more chance* of trouble at sea. It seems a decent way of mimicking reality.

If our algorithm finds that there is going to be no possibility of trouble, the program will go to 1290, the Arrival routine. But if trouble is "expected" we go on to the next lines:

```

985 REM      STORM (990-1022)

990 IF RND (1) > .5 THEN 1030

991 VTAB 13:
  PRINT B$;;
  HTAB 1:
  PRINT " STORM, TAIPAN!":
  GOSUB 760

```

Uh, oh! Looks like we're not having smooth sailing on our maiden voyage. There's now about a 50% chance we'll be sent to the Pirates routine starting at 1030 ("IF RND (1) > .5 THEN 1030"). Remember, line 981 has already determined we're in for trouble, by not sending us to the safety of line 1290. If we didn't get sent off to the Pirates routine right away, now we're headed into a storm.

Let's see if we can ride out this gale:

```

1000 IF RND (1) > .2 THEN 1020

1001 VTAB 14:
  PRINT " ANY PORT IN A STORM,           ":
  GOSUB 760:
  P0 = INT ( RND (1) * 10):
  IF P0 = L THEN VTAB 14:
  PRINT " WE CAN'T MAKE IT,
        TAIPAN,           ":
  SR = 0:
  GOTO 1090

```

In line 1000, we first give the player an 80% chance of riding out the storm unscathed. If this is the case, we GOTO 1020.

If the storm's really a problem, line 1001 will have the ship try to put in at any port at all. We use the random choice of an emergency haven also as a way of finding out if we've survived the storm: if the number of the haven is the same as the number of the port we've just departed, then we didn't make it. (There's no logical justification for this, except the fact that it's a simple and memory-cheap way to do it.) If we "can't make it", we GOTO 1090, where there's a routine that tells us of our fate, and routes us to the That's All Folks routine. (Since line 1090 is also used when we're sunk by pirates, we'll cover that in a later chapter.)

Assuming we've survived the storm, however, the next four lines let us know:

1010 VTAB 14:

```
PRINT " WE'RE HEADED
FOR SHELTER,      ":
GOSUB 780
```

1020 VTAB 14:

```
PRINT " WE RODE OUT THE
STORM, TAIPAN!      ":
GOSUB 780
```

1021 IF RND (1) > .5 THEN 990

1022 GOTO 1290

Line 1021 will, half the time, send us back to line 990, where we may have further troubles. If not, line 1022 sends us directly to the Arrival routine at line 1290.

That's all there is to the Embark routine and the "storm" portion of the Voyage routine.

Some words about how we handled storms: this is one kind of danger which we can never, as players, totally eliminate.

There is always a small chance that in any given voyage, our ship will founder in high seas. The chance is exceedingly small, however. But the Authors feel that it would be unrealistic in the extreme to allow an "unsinkable" ship to sail the simulated seas of the 1860's. Even now, no ship, however huge and powerful, can be considered completely safe from the ravages of oceanic conditions.

The following is a recap of the lines we've introduced in this chapter:

725 REM EMBARK (730-751)

730 IF SH < 0 THEN VTAB 13:

```
PRINT "YOUR SHIP IS OVERLOADED, TAI PAN"      ";;:
PRINT A$:
GOSUB 760:
GOTO 360
```

731 IF SH > = 0 THEN HOME:

```
PRINT TAB( 11)";:
INVERSE:
PRINT "EMBARKING":
NORMAL:
PRINT TAB( 9);"FROM ";L$(L):
INVERSE:
PRINT A$:
NORMAL:
FOR I = 0 TO 9:
IF L = I THEN NEXT I:
GOTO 740
```

732 IF L <> I THEN PRINT TAB(10);I;" ";L\$(I):
NEXT I

```
740 PRINT:  
PRINT:  
INVERSE:  
PRINT A$:  
NORMAL:  
PRINT "WHAT PORT O' CALL, TAIPAN (0-9)?"  
  
750 GOSUB 60:  
IF ASC (X$) > 47 AND ASC (X$) < 58 AND VAL (X$) <>  
L  
THEN P0 = VAL (X$):  
GOTO 980  
  
751 GOSUB 770:  
GOTO 750  
  
975 REM VOYAGE (980-1290)  
  
980 HOME:  
PRINT:  
INVERSE:  
PRINT A$;:  
NORMAL:  
PRINT " SEA VOYAGE FROM ";L$(L):  
PRINT " TO ";L$(P0):  
INVERSE:  
PRINT A$:  
NORMAL:  
GOSUB 780:  
HOME:  
ET = ABS (L0(L) - L0(P0))  
  
981 IF INT ( RND (1) * 50 ) + 1 >  
INT ( RND (1) * ET ) + 1 THEN 1290
```

985 REM STORM (990-1022)

990 IF RND (1) > .5 THEN 1030

991 VTAB 13:

PRINT B\$;:

HTAB 1:

PRINT " STORM, TAIPAN!" :

GOSUB 760

1000 IF RND (1) > .2 THEN 1020

1001 VTAB 14:

PRINT " ANY PORT IN A STORM, " :

GOSUB 760:

PO = INT (RND (1) * 10) :

IF PO = 1 THEN VTAB 14:

PRINT " WE CAN'T MAKE IT,

TAIPAN, " :

SR = 0:

GOTO 1090

1010 VTAB 14:

PRINT " WE'RE HEADED

FOR SHELTER, " :

GOSUB 780

1020 VTAB 14:

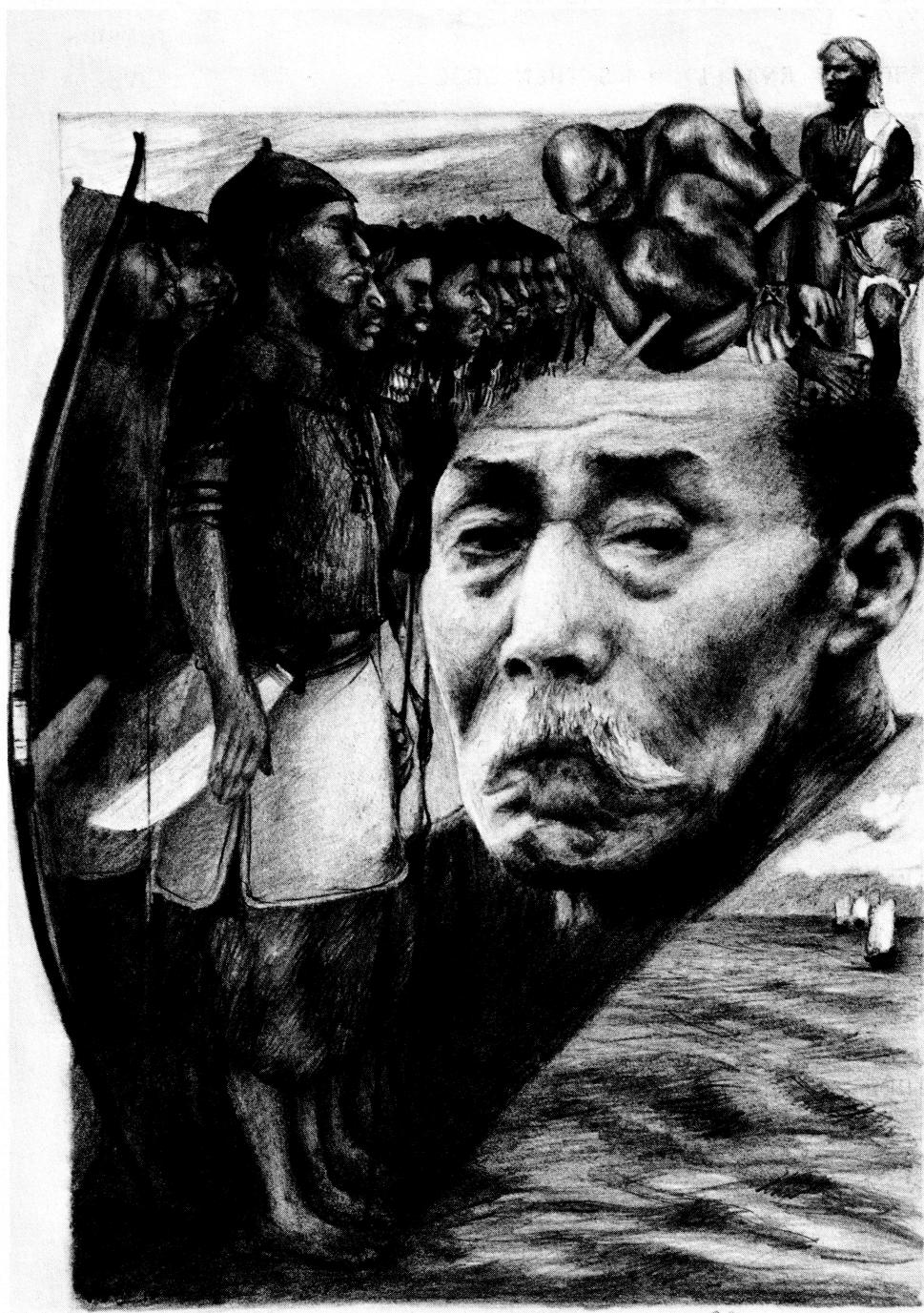
PRINT " WE RODE OUT THE

STORM, TAIPAN! " :

GOSUB 780

1021 IF RND (1) > .5 THEN 990

1022 GOTO 1290



Nicholas Iquan

T A I P A N

A Historical Adventure for the Apple® Computer

Piratical Princes of the Eastern Seas

CHAPTER FOURTEEN

Piracy is defined as the robbery of ships on the high seas.

Along the China Coast, the institution of piracy has a long and fascinating history. Almost all Western piracy occurred during its Golden Age from the mid-seventeenth century to the mid-eighteenth century, and was mainly concentrated in raids against Spanish colonial towns and shipping on the Caribbean and Atlantic. In contrast, the tradition of piracy on the China Seas has covered a much longer period. The pirates of East Asia have operated for over two thousand years in the China Seas, especially in the zone from Hainan island to the Formosa Straits. (Piracy is still something of a problem around Hongkong, though only a part-time occupation of a few mariners.)

What most distinguished the pirates of the China Coast from the buccaneers of the West was their organization and power. Whereas the few hundred buccaneers of the Caribbean usually operated lone ships, and with frequent mutinies among the pirate crews, the East Asian pirates were often organized into massive fleets, with hundreds of thousands of men and thousands of pirate craft.

Without doubt, history's most successful pirate was Kao Hsing-Yeh (1624-1662), better known to the English-speaking by the name "Koxinga", to the Spanish as "Cotsen", and to the Japanese as "Kokusen-ya". To properly understand Koxinga, we must first understand his father.

Koxinga's father was born about 1600 in the small village of Anhai, in China's Fukien province. A poor tailor by trade, the young man who was to become the father of Koxinga headed south from his natal village to seek his fortune in the Portuguese colony of Macau, where he arrived in 1621. There he became a Christian, and was baptized "Gaspar Nicholas".

In Macau this clever young man became better known as Nicholas Iquan. Macau provided the youth with opportunities far beyond those of his home village. He learned Portuguese, then the trade language of East Asia. Soon, Nicholas was to meet a famous Chinese trader, the so-called "Captain China".

Arriving in 1622 at Captain China's Japanese trading headquarters in Hirado, near Nagasaki, Nicholas Iquan went to work for Captain China. He quickly learned to speak Japanese. Nicholas married a Japanese woman, probably of minor samurai family origin. This woman is known to history only by her family name, Tagawa. In 1624, Nicholas Iquan's wife bore a son, who was in Japan called Tagawa Fukumatsu, and raised him for his first six years in Japan, teaching him the Japanese concepts of loyalty and duty.

Nicholas Iquan was soon deeply involved in the far-flung trade — and part-time piracy — of his employer. Captain China himself had been baptized as a Christian, and had escaped from life as a galley slave of the Spanish in the Philippines years before. Captain China had built up a fleet of hundreds of ships, and competed vigorously with the Dutch in the Japan trade, spreading his trading, smuggling and pirating business to countries as distant as Vietnam and Cambodia.

Nicholas Iquan became the most trusted member of Captain China's organization, leading trading voyages involving many ships at a time. In 1624, Nicholas appeared in the Pescadores Islands, in the Formosa Strait, and offered his services as translator to the Dutch, who had their local base there.

As Captain China had urged them to do, the Dutch moved their trading base to Formosa (Taiwan), then a wild and beautiful island inhabited almost entirely by primitive native tribesmen. The Dutch encouraged Chinese settlers to immigrate to Formosa. Nicholas Iquan, using his posi-

tion as a Dutch employee, was able to make a modest extra income from bribes paid to him from the settlers.

Then, in 1625, Captain China died. Nicholas Iquan deserted the Dutch, and moved in on the trader's empire. He killed off supporters of Captain China's legitimate heirs. In a few months, Nicholas controlled the late merchant's four hundred ships. Nicholas Iquan set up his headquarters in his home village of Anhai, and divided his fleet into small, strategically located forces.

By 1627, Nicholas Iquan controlled most of the coast of China, and sacked both Chinese cities and European ships at will. Ships were charged a large annual tribute for safe passage. Portuguese ships were fair game to this mighty pirate, and even his former employers, the Dutch, were targeted. A Dutch punitive fleet of nine men-of-war was sent to destroy his ships at the port of Amoy, but the wily pirate defeated them, and the Dutch squadron fled to Java.

This pirate built a splendid palace for himself at Anhai. His six-year-old son was summoned from Japan to learn the family business, and thus "Tagawa Fukumatsu" was renamed "Cheng Cheng-kung". Nicholas's son became a fine scholar, and enrolled in the Imperial Collegiate School at age fifteen.

Nicholas Iquan surrounded himself with an elite personal guard of three hundred tough and loyal black Africans, former slaves who had escaped Portuguese bondage in Macau.

In 1628, the Ming emperor of China had had enough of this nuisance, and neatly soled his problem by giving Nicholas Iquan the rank of a mandarin, and made him Admiral of the Imperial fleet.

But the Manchus were moving south, bit by bit defeating the weakening Ming dynasty. In 1644, the desperate Ming gave Nicholas Iquan the title, "Count Pacifier of the South", and soon afterward ennobled him as a duke.

Nicholas Iquan was by now joined by his Japanese wife, whom he'd not visited for twenty-one years.

Seeing the fortunes of the Ming declining, Nicholas Iquan became a turncoat. With his elite bodyguard of three hundred exotic and strikingly uniformed blacks, he went to Foochow in 1646 to negotiate the terms of his treason with the advancing Manchus. There he was himself betrayed by the Manchus. After a fierce battle in which more than a third of his

African soldiers died in his defense, Nicholas Iquan was taken captive. He was kept in comfortable house arrest in the Manchu capital of Peking until 1661, when he was beheaded.

Immediately after the surrender of Nicholas Iquan, his wife committed suicide rather than collaborate with the Manchus. This act was not only considered honorable in the Japanese tradition, but it was respected by the Chinese as well.

Now a young man in his early twenties, Cheng Cheng-kung had loved his mother despite their long separation. He burned his scholar's robes, vowing to remain loyal to the failing Ming cause. Cheng gathered a small guerilla band and fought his way to the coast, enlisting his father's former land forces along the way. Within a brief time, he had control of all Nicholas Iquan's land and naval forces. By the time he was twenty-two, Cheng Cheng-kung was a powerful force on both land and sea, controlling the greater part of China's coast and all the China Seas.

The Ming, being defeated everywhere by the Manchus, created Cheng Cheng-kung first an earl, then a marquis, and finally a prince. He opposed the Manchus with vigor, yet looted shipping in the old pirate fashion all the while. In this period, Cheng Cheng-kung was given the honorary name of "Kao Hsing-yeh", which the Dutch have passed on to us as "Koxinga".

Koxinga first centered his activities along the ragged, rugged coasts of China's southeastern provinces of Fukien and Kwangtung. His fleets were said to be able to cover the sea from horizon to horizon. Koxinga's fleets completely dominated the Manchu naval forces, and he carried on massive, devastating raids upon Manchu-held coastal cities of central and south China.

These raids were so troublesome, the Manchu emperor actually had the whole coastal area of the two ravaged provinces (a shoreline over a thousand miles in length) completely evacuated, with the entire population moved more than ten miles inland, on just three day's notice. The manchus had watchtowers, one hundred soldiers manning each tower, set up three miles apart all along the coast. Whole cities, farms, fishing villages and ports were burned to the ground. Half of the evacuees died of starvation, and many of the others were sold into slavery. Anyone found near the coast was promptly beheaded without trial.

This bizarre and unique solution was enforced for nineteen years, by Imperial edict.

Koxinga turned in frustration to ousting the Dutch from their base at Fort Zeelandia on Formosa, invading the island in force.

The Dutch were widely known for their tenacity in holding their factory posts against heavy odds. But the Dutch governor of Formosa, Frederik Coyett, had not been given the support he'd requested from the colonial authorities in Batavia. Though able to hold out for nine bitter and hungry months against thirty-to-one odds, Coyett eventually surrendered to Koxinga in 1661. Koxinga treated the defeated Dutch with honor and mercy, allowing them to withdraw in safety from the island. The evening of the Dutch surrender, Koxinga pronounced himself King of Formosa.

Only a bit more than two hundred miles south of Formosa lay the Philippines. There, Chinese residents had for many years undergone persecution and periodic genocide. Koxinga felt protective toward these Chinese, and he sent word to the Spanish in Manila that they must either pay him tribute, or be invaded. The Spanish refused, and, unfairly assuming that all the Chinese living in Manila were conspiring with Koxinga, they massacred the local Chinese. This genocidal attack enraged the pirate king. Koxinga was planning his conquest of the Philippines when he died, maddened by a fever, on July 2, 1662, at the age of thirty-seven.

In 1700, the Manchus, having conquered China, officially praised Koxinga, their deceased enemy. Now in the position of an established dynasty, the Manchu emperor lauded Koxinga as loyal dynastic defender, urging his subjects to follow the pirate's example. In 1875, the Manchus actually *deified* Koxinga as the God of Loyalty and Fidelity! The Japanese in 1898 inducted Koxinga as a Shinto god, while using his half-Japanese heritage as a pretext for their occupation of Formosa.

Since the time of Koxinga, piracy has never again been as enormous an enterprise. Yet it has persisted, with major surges during times of disruption. By 1800, it is said that there were two squadrons of pirate ships operating off the South China coast, together totaling more than 600 junks and 60,000 pirates of international origin. The Opium Wars, and the disorder caused by them, caused another increase in piracy in the 1840s.

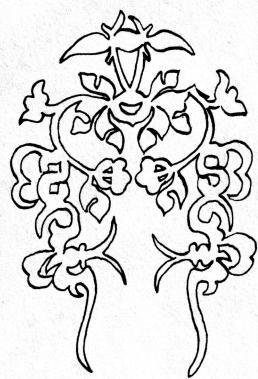
In the game of *Taipan*, piracy as a factor is exaggerated for game purposes. Yet pirates were still a serious consideration in the 1860s. Nearly every Chinese or foreign cargo ship plying trade on the China seas car-

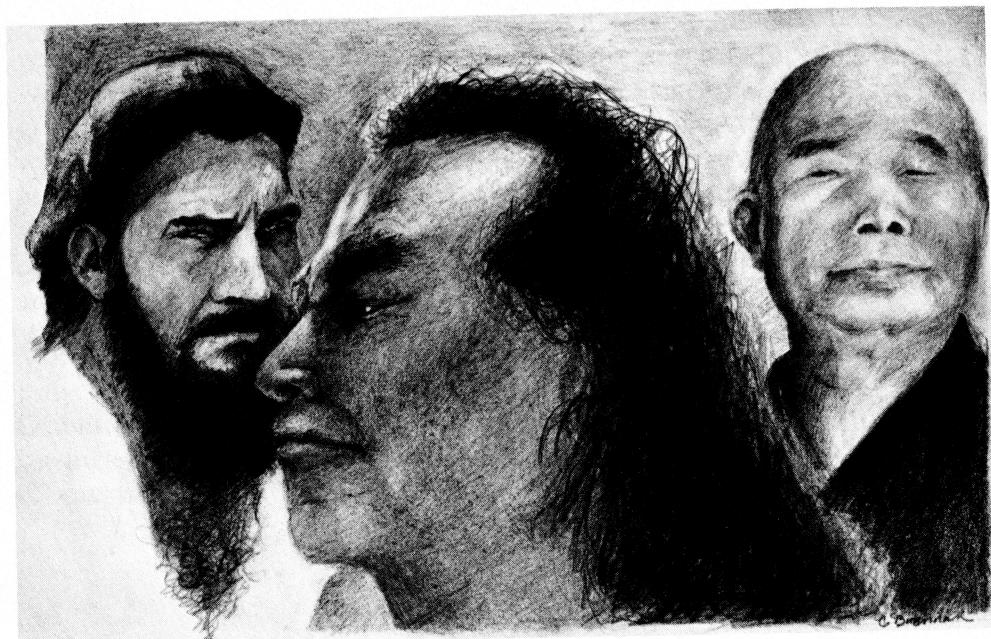
ried either jingals (swivel-mounted blunderbusses about seven feet long and weighing only twelve pounds) or Western cannons, for use against pirates.

Crews carried axes, spears, pistols and rifles with which to repel boarders. Bullet-proof shields to protect helmsmen were standard equipment on seagoing junks. Even the great clipper ships would sometimes fall prey to oar-driven pirate junks, should they find themselves becalmed.

With the advent of steamships, craft invulnerable to being boarded in windless seas, pirates found new ways of preying on shipping. Pirates would board a steam vessel in port, disguised as normal paying passengers. When the ship was at sea, and the time seemed ripe, the pirates would pull out guns, knives, hatchets and clubs, taking over the bridge and the engine room. The passengers, crew and cargo would be robbed, rich passengers kidnapped for ransom, and the pirates and their material and human booty would disappear overboard into waiting "fishing" craft, which had rendezvoused by pre-arrangement.

In our game of *Taipan*, we assume that the (fictional) pirate Li Yuen is powerful, but not nearly so strong as some of the historic pirates of the Orient. Thus he is not able to fully control his competitors in the piracy business. Still, a payment of tribute to Li is worthwhile just to keep him off our backs, and he may be able to protect us from rival pirate gangs.





Smythe

Li Yuen

Yamato

T A I P A N

A Historical Adventure for the Apple® Computer

Stand by the Swivel Guns!

CHAPTER FIFTEEN

These lines are the Pirates routine:

```
1025 REM      PIRATES (1030-1281)

1030 HOME:
      ID = INT ( RND (1) * 2 ) + 1:
      GOSUB 5290:
      P = INT ( RND (1) * (MW / 25) * 2 ^ (ID * 2) )
      + INT ( RND (1) * 3 ) + 1:
      HTAB 1:
      VTAB 17:
      PRINT P;" PIRATE CRAFT SIGHTED!":
      GOSUB 780:
      GOSUB 1100
```

Line 1030 determines how many pirates will be encountered, and whether they are part of Li Yuen's fleet or independents.

(With GOSUB 5290, it also starts the Sea Action subroutines, to make the pirates appear on the screen. At the end of the book, we'll cover all of the Sea Action subroutines, which have line numbers of from 5000 to 6360. We'll largely ignore these GOSUBs for the moment.)

Remember that it's very likely that this line, like those in the Storm routine, won't be used at all in any particular voyage, as line 981 may have passed it by completely. If we do see pirates, there'll be a 50% chance they will be either Li Yuen's gang or independent scoundrels. The variable "ID" is used to flag the identity of the pirates: a value of 1 stands for independents, and 2 represents Li Yuen's jolly crew. The second part of line 1030 is a complicated formula which determines P, the number of pirate craft encountered. This number is based upon the size of our vessel (MW, the maximum cargo capacity of the ship), whether or not we've met Li Yuen's bully boys (they normally travel in much larger fleets), and a random factor.

The rest of this line simply tells us that we've sighted "P" number of pirate ships, and GOSUBs 1100, where the Ship Status subroutine starts. Here's the Ship Status subroutine:

```
1095 REM      SHIP STATUS SUBROUTINE (1100-1110)

1100 HTAB 1:
      VTAB 16:
      INVERSE:
      PRINT A$;;
      NORMAL:
      PRINT " GUNS: ";G;"          REPAIR:";:
      IF SR < 0 THEN SR = 0

      1110 PRINT INT (SR * 100);%"  ":
      PRINT "SHIPS ENCOUNTERED:";P;"      ":
```

```

PRINT A$;;
PRINT A$;;
PRINT A$;;
RETURN

```

This subroutine shows us how the battle, if any, is going for us. We'll be calling this subroutine from several points in the Pirates routine. It displays the number of guns with which our craft is armed, our ship's state of repair (which must stay above 10% for us to stay above the surface of the deep blue sea), and shows the number of pirate ships encountered at any time.

The variable "SR" is a number which will never be more than 1, and keeps track of the state of repair of our ship. Line 1100 makes sure that the value of SR never is below zero before being displayed. In line 1110, the state of repair is displayed as SR times 100, in order to give us the state of repair as a percentage figure. We then RETURN to line 1030.

The next lines in the Pirate routine are 1040 and 1041:

```

1040 IF ID = 1 THEN VTAB 19:
PRINT " LOOKS LIKE INDEPENDENT PIRATES.":
GOSUB 780

1041 IF ID <> 1 THEN VTAB 19:
PRINT B$;;
HTAB 1:
PRINT " THERE'S ";LY$;"'S BANNER!":
GOSUB 780

```

All we're doing in these lines is displaying the identity of the pirates.

Line 1050 handles the possibility that we've not only encountered independent pirates, but that Li Yuen's fleet has come up and driven off the "poachers":

```

1050 GOSUB 1100:
  IF ID = 1 AND RND (1) > .95
  THEN VTAB 19:
  PRINT B$;;
  HTAB 1:
  PRINT " ";LY$;"'S FLEET DROVE'EM OFF!":
  GOSUB 6100:
  P = P * INT ( RND (1) * 10) + 6:
  ID = 2:
  GOSUB 5290:
  GOSUB 1100:
  IF TR = 0 THEN 1070

```

When this line operates, independent pirates will be chased away by Li Yuen's fleet 5% of the time. But since we'll be looping back to this line probably several times during any pirate encounter, there's a real good chance that Li Yuen's fleet will appear to banish the interlopers. Notice that 1050 sets the number of ships in Li Yuen's fleet as being the integer of "RND (1) * 10" multiplied by the number of independent pirates, then adds another six. In other words, Li's fleet will be anywhere from one times the size of the independent fleet (plus five), to ten times (plus five).

After calling the appropriate subroutine in the Sea Action section of the program (GOSUB 6100), line 1050 then gives us a Ship Status update, sets the ID flag to 2, to indicate Li Yuen's fleet, and checks whether we're "paid up" with Li Yuen's favorite charity. If we're not in good standing with Li (TR = 0), we go to line 1070.

Lines 1060 and 1061 only operate if the pirate fleet encountered is that of Li, and we're paid up with that gentleman:

```

1060 IF ID = 2 AND TR = 1 THEN VTAB 19:
  PRINT "THEY GREET US, AND SAIL OFF.    ":
  GOSUB 6100:
  P = 0:
  GOSUB 780:
  IF RND (1) > .8 THEN 1030

```

```
1061 IF ID = 2 AND TR = 1 THEN 1290
```

As we can see, if the pirates on hand are Li's fellows (ID = 2), and we've paid tribute recently enough (TR = 1), the pirate fleet departs peacefully (P = 0). Then, twenty percent of the time, this line starts looking for the possibility of further pirate troubles, by looping back to 1030. If we luck out, however, the program branches from 1061 straight to the Arrival routine at 1290.

The *much* less pleasant possibility is that we are in bad stead with Li Yuen, either by refusing to help him build his temple, or by being so unfortunate as to be caught with an "expired" contribution. (How such a contribution can expire will be explained when we later cover the Arrival routine and the Update subroutine.) For whatever reason, we're in trouble with Li Yuen now.

Line 1070 shows us that Li's minions, Yamato and Smythe, are less than pleased to meet us:

```
1070 IF CR = 0 AND TR = 0
      AND ID = 2 THEN CR = 1:
      VTAB 19:
      PRINT B$;;
      HTAB 1:
      PRINT " ";YS$;" ARE CURSING US!":
      GOSUB 780
```

Note the use of the flag CR, which is used for no other purpose than to make certain that a well-worded curse isn't over-used each time the program goes past this line. One curse by the likes of Yamato and Smythe is enough to last a voyage.

The following lines demonstrate that Mssrs. Yamato and Smythe are capable of more than mere words:

```
1080 VTAB 19:
      PRINT " THEY'RE FIRING ON US!
      GOSUB 5380
```

```

1081 IF INT ( RND (1) * P) + 1 >
    INT ( RND (1) * 5) + 1 THEN GOSUB 5540:
    VTAB 19:
    PRINT B$;;
    HTAB 1:
    PRINT " THE BUGGERS HIT US!":
    SR = SR - ( RND (1) /
    (MW / ( INT ( RND (1) * 50) + 1)))):
    GOSUB 780:
    GOSUB 1100:
    GOTO 1090

1082 VTAB 19:
    PRINT " MISSED US, TAIPAN!      ":
    GOSUB 780

```

Holey waterline! This is getting serious. These lines handle the gunfire of not only Li's fleet, but that of independents, as well. The "IF INT (RND (1) * P) + 1 > INT (RND (1) * 5) + 1 THEN..." part is used as a method of making it more likely that a large attacking fleet will hit us than would a smaller fleet, during any given salvo. The figure of "5" is arbitrary, arrived at through trial and error, as a reasonable factor to help give us the accuracy of the attackers. An attacking force of more than five will *usually* hit us during a salvo, and a smaller force will *usually* miss us.

Next, line 1081 determines the damage done if we're hit. SR, we'll recall, represents the state of repair of our ship. When we leave port, this figure is 1, and any damage done to our ship makes SR a decimal fraction. Damage is computed in line 1081 by subtracting a decimal fraction from SR. The fraction to be subtracted from our ship's state of repair is determined by first dividing the figure MW (the maximum capacity, or size, of our ship) by an integer between one and fifty. After this first division is made the resulting figure is used to divide "RND (1)" (a random fraction), and the result of *that* division gives us a fraction (usually a small one), which is subtracted from our state of repair, to give us a smaller value for SR.

The purpose of this fairly complex formula is simply to assure that if we have a large ship, we're likely to be less damaged by a single salvo than would a smaller ship.

After the damage is computed, line 1081 then GOSUBs 1100 to display the new Ship Status. But if the pirate salvo missed us, that delightful information is supplied to us instead by line 1082.

Now come even more ominous program lines:

```
1090 IF SR < .1 THEN GOSUB 780:
  VTAB 19:
  PRINT " WE'RE GOING UNDER, TAIPAN!      ":
  GOSUB 6290:
  X$ = " ":
  GOTO 1300

1091 GOTO 1120
```

These are lines we promised to cover later, back when we went through the Storm routine. Before we went to line 1090 from the Storm routine, we'd set SR to 0.

For whatever reason our ship is now sinking, we'll always make an extended visit to Davy Jones' Locker Club if our state of repair (SR) is less than .1 (10% state of repair when displayed on the screen by the Ship Status subroutine).

X\$ is made to hold a null string (nothing), in order that the That's All Folks routine starting at 1300 will never see it holding an "R", which would have been the case if we'd pressed "R" for Retire to reach 1300 from the Market routine. (We don't want to be given a chance to continue where we "left off", if we're really supposed to be goners!)

If we didn't sink this time, we move along to lines 1120, 1121, 1122 and 1123:

```

1120 VTAB 19:
  PRINT " SHALL WE R)UN, F)IGHT, OR P)ARLEY?":
  GOSUB 60:
  IF X$ = "R" THEN X = 1

1121 IF X$ = "F" THEN X = 2

1122 IF X$ = "P" THEN X = 3

1123 IF X$ <> "R" AND X$ <> "F" AND X$ <> "P"
  THEN GOSUB 770:
  GOTO 1120

```

At last, we'll get to do something about these dratted pirates!

After line 1120 asks us whether we wanted to Run, Fight or Parley, we are left with the value of 1, 2, or 3 in the flag X.

Line 1130 routes the program flow to the appropriate lines:

```
1130 ON X GOTO 1140,1160,1200
```

If our choice was to Run, the value of variable X would be 1, and we'd GOTO line 1140:

```

1140 IF SR <= RND (1) AND INT ( RND (1) * 6 ) + 1
  <= INT ( RND (1) * P ) + 1 THEN 1150

1141 IF SR > RND (1) AND INT ( RND (1) * 6 ) + 1
  > INT ( RND (1) * P ) + 1 THEN 1143

1142 GOTO 1150

1143 GOSUB 6220:
  VTAB 18:
  PRINT " WE ESCAPED FROM 'EM TAIPAN! "

```

```

P = 0:
GOSUB 780:
GOSUB 1100:
IF RND (1) > .8 THEN 1030

1144 GOTO 1290

```

The variable SR, of course, is the state of repair of our ship. It is always zero and one, inclusively, and it indicates the general condition of our vessel. Line 1140 first takes SR and compares it to a random fraction. If SR is less than or equal to RND (1), and RND (1) * 6 (plus one) is also less than or equal to RND (1) * P (plus one), then we haven't escaped the pirates this round ("THEN 1150"). More simply put: there are two factors which increase the player's chances of getting away from the pirates: a high state of repair (SR), and a low number of pirates (P).

If we've escaped, we're routed to line 1143, where a message to this effect is displayed. The number of pirates is reduced to zero (P=0), and the Ship Status subroutine is run ("GOSUB 1100"). But there's always a chance that we'll see more "sea trouble", so two times out of ten, we'll be routed back to 1030, where we'll have another chance to get wiped out. If there is not further chance of storm or pirates, line 1144 routes us to the Arrival routine (1290), and we're home free, filled with noble sentiments about discretion and valor.

But if we haven't been able to run away from *all* of the pirate ships, lines 1150 and 1151 handle the other two options:

```

1150 IF SR > RND (1) AND RND (1) > .6 THEN
  P = P - INT ( INT (( RND (1) * P ) + 1) / 2):
  GOSUB 6220:
  GOSUB 5290:
  VTAB 19:
  HTAB 1:
  PRINT P;" OF 'EM STILL WITH US!
  ":
  GOSUB 780:
  GOSUB 1100:
  GOTO 1050

```

```

1151 VTAB 19:
  PRINT " CAN'T SHAKE 'EM!
  ":
  GOSUB 780:
  GOTO 1050:

```

These lines take care of either the possibility that we got away from *part* of the pirates (line 1150), or that we couldn't lose *any* of them (line 1151). The complicated first part of line 1150 bases our chances of escaping some of the pirate fleet partially upon our state of repair. And the number of pirate ships still with us, should we be so lucky, is set to be at least half the original number. Like many of the routines in Taipan, we here have to carefully balance ease versus difficulty. The chance of losing the pirates on any one run-through is very low. But we'll be going through these lines as long as our ship is afloat, and still trying to escape. So we *want* the chance of escape to be quite low each time these lines are run.

Here are the lines we've waded through in this chapter:

```

1025 REM      PIRATES (1030-1281)

1030 HOME:
  ID = INT ( RND (1) * 2) + 1:
  GOSUB 5290:
  P = INT ( RND (1) * (MW / 25) * 2 ^ (ID * 2))
  + INT ( RND (1) * 3) + 1:
  HTAB 1:
  VTAB 17:
  PRINT P;" PIRATE CRAFT SIGHTED!":
  GOSUB 780:
  GOSUB 1100

1040 IF ID = 1 THEN VTAB 19:

```

```
PRINT " LOOKS LIKE INDEPENDENT PIRATES.":  
GOSUB 780

1041 IF ID <> 1 THEN VTAB 19:  
PRINT B$;:  
HTAB 1:  
PRINT " THERE'S ";LY$;"'S BANNER!":  
GOSUB 780

1050 GOSUB 1100:  
IF ID = 1 AND RND (1) > .95  
THEN VTAB 19:  
PRINT B$;:  
HTAB 1:  
PRINT " ";LY$;"'S FLEET DROVE'EM OFF!":  
GOSUB 6100:  
P = P * INT ( RND (1) * 10) + 6:  
ID = 2:  
GOSUB 5290:  
GOSUB 1100:  
IF TR = 0 THEN 1070

1060 IF ID = 2 AND TR = 1 THEN VTAB 19:  
PRINT "THEY GREET US, AND SAIL OFF.":  
GOSUB 6100:  
P = 0:  
GOSUB 780:  
IF RND (1) > .8 THEN 1030

1061 IF ID = 2 AND TR = 1 THEN 1290
```

```
1070 IF CR = 0 AND TR = 0
      AND ID = 2 THEN CR = 1:
      VTAB 19:
      PRINT B$;;
      HTAB 1:
      PRINT " ";YS$;" ARE CURSING US!":
      GOSUB 780

1080 VTAB 19:
      PRINT " THEY'RE FIRING ON US!           ":
      GOSUB 5380

1081 IF INT ( RND (1) * P) + 1 >
      INT ( RND (1) * 5) + 1 THEN GOSUB 5540:
      VTAB 19:
      PRINT B$;;
      HTAB 1:
      PRINT " THE BUGGERS HIT US!":
      SR = SR - ( RND (1) /
      (MW / ( INT ( RND (1) * 50) + 1))): 
      GOSUB 780:
      GOSUB 1100:
      GOTO 1090

1082 VTAB 19:
      PRINT " MISSED US, TAIPAN!           ":
      GOSUB 780

1090 IF SR < .1 THEN GOSUB 780:
      VTAB 19:
      PRINT " WE'RE GOING UNDER, TAIPAN!           ":
      GOSUB 6290:
      X$ = " ":
      GOTO 1300
```

1091 GOTO 1120

1095 REM SHIP STATUS SUBROUTINE (1100-1110)

1100 HTAB 1:

VTAB 16:

INVERSE:

PRINT A\$::

NORMAL:

PRINT " GUNS: ";G;" REPAIR: ";

IF SR < 0 THEN SR = 0

1110 PRINT INT (SR * 100);%" "

PRINT "SHIPS ENCOUNTERED: ";P;" "

PRINT A\$::

PRINT A\$::

PRINT A\$::

RETURN

1120 VTAB 19:

PRINT " SHALL WE R)UN, F)IGHT, OR P)ARLEY?":

GOSUB 60:

IF X\$ = "R" THEN X = 1

1121 IF X\$ = "F" THEN X = 2

1122 IF X\$ = "P" THEN X = 3

1123 IF X\$ <> "R" AND X\$ <> "F" AND X\$ <> "P"
THEN GOSUB 770:
GOTO 1120

1130 ON X GOTO 1140,1160,1200

1140 IF SR < = RND (1) AND INT (RND (1) * 6) + 1
< = INT (RND (1) * P) + 1 THEN 1150

1141 IF SR > RND (1) AND INT (RND (1) * 6) + 1
> INT (RND (1) * P) + 1 THEN 1143

1142 GOTO 1150

1143 GOSUB 6220:

VTAB 18:

PRINT " WE ESCAPED FROM 'EM TAIPAN! "

P = 0:

GOSUB 780:

GOSUB 1100:

IF RND (1) > .8 THEN 1030

1144 GOTO 1290

1150 IF SR > RND (1) AND RND (1) > .6 THEN
P = P - INT (INT ((RND (1) * P) + 1) / 2):

GOSUB 6220:

GOSUB 5290:

VTAB 19:

HTAB 1:

PRINT P;" OF 'EM STILL WITH US!

" :

GOSUB 780:

GOSUB 1100:

GOTO 1050

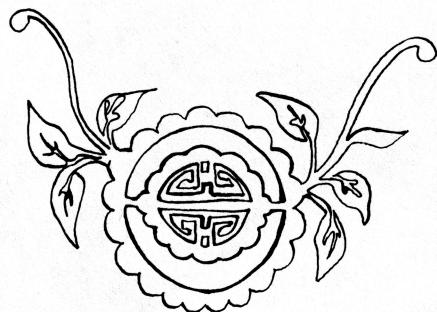
1151 VTAB 19:

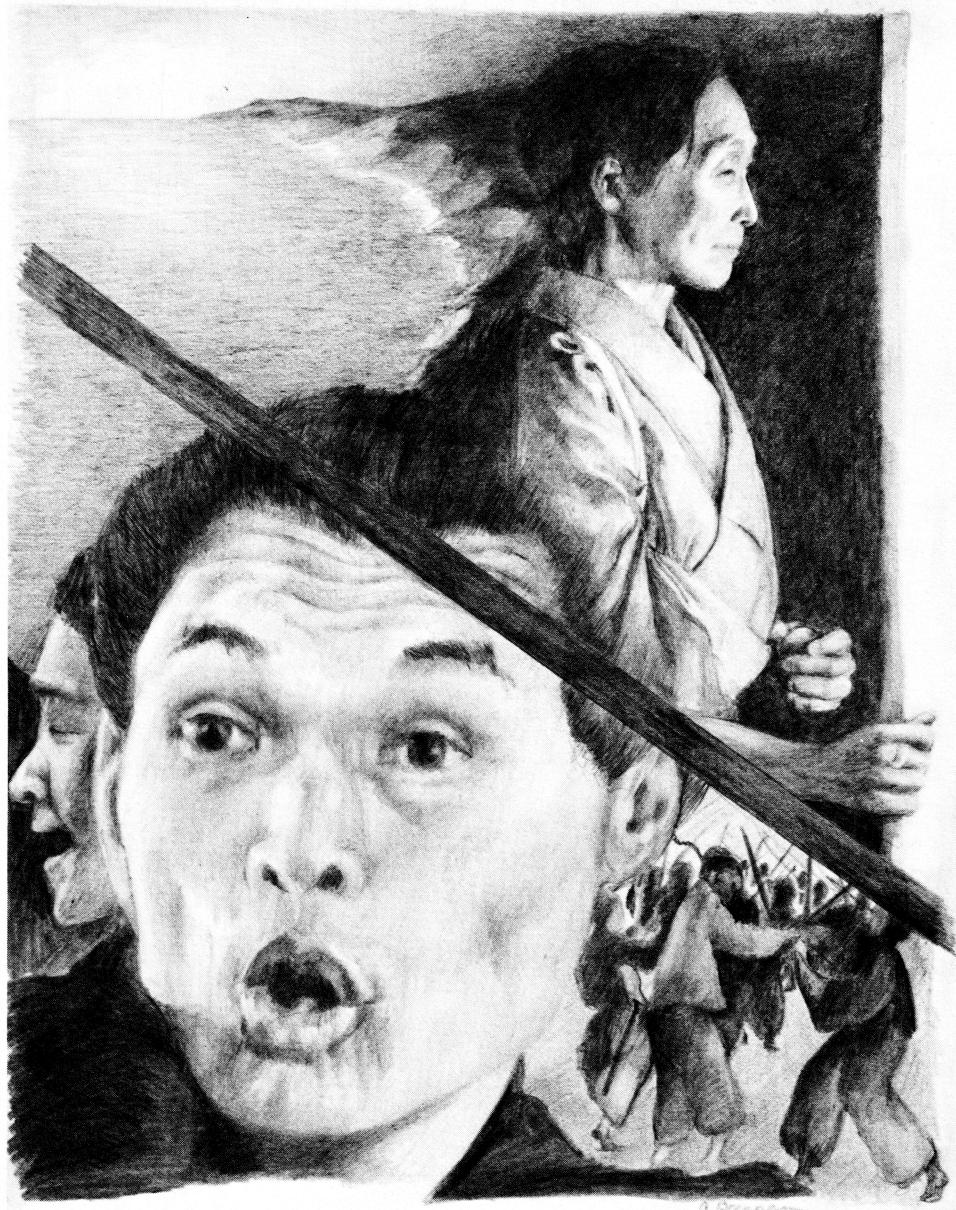
PRINT " CAN'T SHAKE 'EM!

" :

GOSUB 780:

GOTO 1050:





Koxinga

T A I P A N

A Historical Adventure for the Apple® Computer

More Pirates

CHAPTER SIXTEEN

The "Run" choice sends us to lines 1160 and 1161:

1160 VTAB 19:

```
PRINT " WE'RE FIRING ON 'EM! ":
GOSUB 5540:
IF INT ( RND (1) * (G + 1)) + 1 > INT (RND (1) *
4)
THEN VTAB 19:
PRINT B$;;
HTAB 1:
PRINT "HIT 'EM!":
GOSUB 5680:
GOTO 1170
```

1161 VTAB 19:

```

PRINT " MISSED 'EM!
":  

GOSUB 780:  

GOTO 1190

```

Here the factor which tunes the accuracy of our shots at the pirates is the *number of guns* we have. The more guns, the more likely we are to hit pirates.

The next lines determine the effect of our hits on the pirates:

```

1170 NK = RND (1):  

  IF NK > .4 THEN IF P > = G THEN  

    X = INT ( INT (( RND (1) * (G - 1)) + 1) / 3) + 1  

1171 IF NK > .4 AND P < G THEN X = INT ( RND (1) * P)  

  + 1  

1172 IF NK < = .4 THEN VTAB 19:  

  PRINT " THEY'RE STILL AFLOAT.":  

  GOSUB 780:  

  GOTO 1190

```

Line 1170 sets a value to variable "NK", which will, with other factors, determine the effect of our guns: a value of more than .4 is needed for *any* pirate craft to be sunk. If none were, line 1172 operates, and we're given the display which tells us we haven't sunk any ships. Otherwise lines 1170 and 1171 have to figure out *how many* pirate craft we've sunk. They do this by first figuring if there are at least as many pirate ships as we have guns (. . . IF P > = G THEN X = INT (INT ((RND (1) * (G - 1)) + 1) / 3) + 1 . . .). If there are pirate ships in numbers greater than or equal to the number of guns we have, then, clearly, we can't possibly sink them all with one salvo. This portion of line 1170 sets X to the number actually sunk by our fire.

If there are *less* than one pirate craft for each of our guns, line 1171 sets X to a random number between one and the total number of pirate ships (. . . X = INT (RND (1) * P) + 1 . . .).

The victims of our blazing gunfire are noted and subtracted from the total number of pirate ships in 1180:

```
1180 P = P - X:
  VTAB 19:
  PRINT " SANK ";X;" OF 'EM, TAIPAN!":
  GOSUB 5830:
  IF P > 0 THEN GOSUB 5290
```

We must now determine whether there are any pirate vessels left. Lines 1190 through 1193 do this for us:

```
1190 GOSUB 1100:
  IF P = 0 THEN IF RND (1) > .5 THEN 1270

1191 IF P = 0 THEN VTAB 19:
  PRINT " THAT'S ALL FOR THE BUGGERS!":
  GOSUB 780:
  IF RND (1) > .8 THEN 1030

1192 IF P = 0 THEN 1290

1193 GOSUB 780:
  GOTO 1050
```

The first thing line 1190 does is to have the Ship Status subroutine (1100) update the screen.

There is now a fifty-fifty chance that the program will go to 1270, where the Capture routine is located. (This Capture routine, which we'll describe soon, allows us to take control of a larger ship.) If no pirate ship is captured, a display in line 1191 tells the player that the pirates are done for. Then there is a one-out-of-five chance that line 1191 recycles us to 1030 for more pirates. Otherwise, line 1192 sends us directly to the Arrival routine at 1290.

If we haven't defeated the pirates at all, line 1193 has us go back to 1050 to continue our confrontation.

Supposing we'd made a choice to Parley with the pirates, we'd be sent by line 1130 to line 1200:

```
1200 IF RND (1) < .8 THEN VTAB 19:
  PRINT " THEY REFUSE TO PARLEY, TAIPAN.      ":
  GOSUB 780:
  GOTO 1050
```

```
1201 VTAB 19:
  PRINT " THEY AGREE TO DISCUSS TERMS.      ":
  GOSUB 780
```

Each time we try to parley, there's about an 80% chance that the pirates will have none of it, and they'll get a free shot at us in the bargain, when we GOTO 1050. If they do agree to talk (line 1201), we continue to line 1210:

```
1210 IF P > G / 2 OR RND (1) > .7 THEN VTAB 19:
  PRINT " THEY OFFER TO LET US GO IN EXCHANGE FOR"::
  PRINT "ALL OUR CASH. DO WE ACCEPT, TAIPAN (Y/N)"::
  GOTO 1220
```

```
1211 GOTO 1240
```

Such a *deal!* Of course, the player must decide if this is an offer that can be refused. If we have almost all our money invested in trade goods, "your money or your life" may sound pretty attractive. But if we're dead-heading it back from Shanghai with an empty hold and three million in cash from a big opium sale, we'd be a bit less interested — we might want to put our lives on the line to protect that money.

The following lines process the player's decision:

```

1220 GOSUB 60:
  VTAB 19:
  PRINT A$;;
  PRINT A$;;
  VTAB 18:
  IF X$ = "Y" THEN IF RND (1) > .2 AND
  C > 100 + INT ( RND (1) * 2000) THEN P = 0:
  C = 0:
  VTAB 19:
  PRINT " THEY TOOK IT & RAN!":
  GOSUB 780:
  GOSUB 1100:
  GOTO 1280

```

```
1221 IF X$ <> "Y" THEN 1050
```

```

1222 C = 0:
  VTAB 19:
  PRINT " THEY STILL INTEND TO FIGHT!":
  PRINT:
  PRINT A$;;
  GOSUB 780

```

The first thing line 1220 does is to GOSUB 60, which gives us either a "Y" held in X\$, or any other keyboard character. For a simple decision like this, we'll just consider *anything but* a "Y" to be a negative reply. But simple though the decision may be, the handling is a bit complex.

To make the game more like the real world, it's best that the *result* of giving in to the pirates' demands not be certain. These lines make it uncer-

tain by first giving about a 20% chance that the pirates will take our money and *still* continue attacking, then by checking whether our cash is greater than an arbitrary random figure between 100 and 2100. Thus, if the cash we offer the pirates for our safety is less than that random figure, once again we'll continue to suffer the pirate attack. The amount of money the pirates will consider enough will vary from time to time. The pirates may consider our cash insufficient funds and refuse to leave us alone. But they'll keep our money even while continuing their attack.

If the pirates consider our cash fair pay for the work they've done, line 1220 will GOTO 1280, at the end of the Pirate routine, and we're left poorer but safer.

In two cases we'll get to line 1230: if we refused to fork over our money, or if they took it and want our hides, as well.

```
1230 GOSUB 780:
      GOTO 1050
```

All this line does is to give a delay (GOSUB 780) to allow us to read the last message, and route us back to the midst of battle, at 1050.

If there are less pirate vessels than the number of our guns divided by two, and the "RND (1) > .7" part of line 1210 is "false", then the *pirates* will try to give up:

```
1240 VTAB 1:
      HTAB 13:
      PRINT $FL$;;
      VTAB 19:
      PRINT " THEY OFFER TO SURRENDER.      ":
      GOSUB 780:
      VTAB 19:
      PRINT " DO WE LET THEM GIVE UP (Y/N)?      ":
      GOSUB 60:
      IF X$ = "Y" THEN 1260
```

Now we have a choice of continuing the battle, or allowing the pirates to surrender. The last part of line 240 routes us to line 1260 if we accept

the surrender, otherwise (if we didn't accept the surrender) the program flow falls through to line 1250, where there's a chance that a number of pirate vessels will run off:

```
1250 IF RND (1) > .5 AND P > 0
  THEN E = INT ( RND (1) * P ) + 1:
  P = P - E:
  GOSUB 6100:
  VTAB 19:
  PRINT B$;;
  HTAB 1:
  PRINT E;" OF 'EM RAN AWAY!":
  GOSUB 780:
  GOSUB 1100:
  GOSUB 5290:
  GOTO 1050
```

This line will *half the time* allow a number between *one* and *all* of the pirate ships to flee, and will use the temporary variable E to indicate how many got away. If one or more got away, we go back to line 1050 for another cycle through the battle routine. If none escaped, we fall through to line 1260:

```
1260 IF RND (1) > .7 THEN VTAB 19:
  PRINT " THEY'RE PREPARING TO ATTACK!    ":
  GOSUB 780:
  GOTO 1050
```

So, even if the pirates have asked to surrender, and we've accepted their offer, they might attack anyway. What we're trying to do here is to account for some of the rich mixture of happenings which are found in the real world of human beings. *Nothing* is certain when we're dealing with people and the authors have designed Taipan with this in mind. If the pirates betray us by attacking now, the program goes back to 1050 to continue the carnage.

If the pirates are sincere in their surrender, the next line allows us to take control of a captured ship, superior in size and armament to our own:

1265 REM CAPTURE (1270-1281)

1270 VTAB 19:

```
PRINT " WE'VE CAPTURED A BIGGER SHIP!":
GOSUB 760:
VTAB 19:
PRINT " WE'RE TRANSFERRING TO IT NOW.":
GOSUB 780:
G = G + INT ( RND (1) * (G + 1)) + 1:
E = SH + INT ( RND (1) * (SH + 150)) + 1:
SH = SH + E:
MW = MW + E:
P = 0
```

The number of guns (G) we'll have on the new ship is determined by "G=G+INT (RND (1) * (G+1))+1", which gives us at least one more than the number we had before, and as many as twice-plus-one the number we had previously. The total cargo capacity of our ship's hold (SH) is increased by the statements, "E=SH+INT (RND (1) * (SH+150))+1" and "SH=SH+E", which make the available capacity of our new ship's hold at least as large as our old one (plus one unit), and possibly as large as double the original (plus one hundred and fifty units). "MW=MW+E" adds the same increase on to the *total* capacity of our hold (MW).

The next lines decide whether we've got more combat, we're being sunk, or we're finally free of pirates and ready to enter port:

1280 IF SR < .1 THEN 1300

1281 IF P > 0 THEN 1050

Line 1280 checks if we have a state of repair (SR) of less than a tenth (or 10%). If that's true, it gets rid of our sinking hulk by hurling us into the maelstrom of the That's All Folks routine at line 1300. Line 1281 simply checks whether or not there are any pirate vessels left. If there are, we must once again recycle through the combat routine, starting again at line 1050.

If neither of these conditions are true, we fall through to line 1290, the Arrival routine, which is the last part of the Voyage routine:

1285 REM ARRIVAL (1290)

```
1290 CR = 0:  
SR = 1:  
L = P0:  
V(L) = V(L) + 1:  
B = 0:  
K = 0:  
GOSUB 160:  
HOME:  
PRINT:  
INVERSE:  
PRINT A$;  
NORMAL:  
PRINT " ARRIVING ";L$(L);" AFTER":  
PRINT " A VOYAGE OF ";GET;" DAYS.":  
INVERSE:  
PRINT A$;  
NORMAL:  
GOSUB 780:  
HOME:  
GOTO 120
```

A whole slew of things have to be done at this point. First, CR, the flag which indicates whether Yamato and Smythe have cursed us this time at sea, is reset to zero. Next, SR is set to one, to indicate that any damage to our ship is now repaired. L, the identity of the port we're visiting, is made to equal PO, our chosen port o' call. (Of course, if a storm forced us into a port for shelter, we didn't actually *choose* the value in PO.)

Next, the number of times we've visited the port we're entering is incremented by adding one to V(L). (Remember? We use the V(L) array in the Records subroutine.)

The flag B, which indicates whether we've already borrowed from Elder Brother Wu during our present time in port, is reset to zero, to show we haven't done so. (This would only be important if we're in Hongkong.) Yet another flag, K, is reset to zero to show that the Events routine has not yet been run through for this port. (We'll discuss that routine in a separate chapter.)

Line 1290 next GOSUBs 160. Lines 160 to 172 are the Update subroutine. Since they need to be explained now, here they are:

```

155 REM UPDATE SUBROUTINE (160-172)

160 ET = INT (ET + (ET * RND (1) / 3)):
  GT = GT + ET:
  D = D + INT (D * (ET / 360)):
  JD = JD + ET:
  IF JD > 360 THEN JD = JD - 360:
  Y = Y + 1

161 IF JD < 1 THEN JD = 1

170 M = INT ((JD - 30) / 30):
  DA = INT (((((JD / 30) - INT (JD / 30))) * 30):
  IF RND (1) > .95 THEN TR = 0

171 IF M < 0 THEN M = 0

```

172 RETURN

Except for the tail-end of this subroutine, all it does is calculate the date after an ocean voyage, by adding ET (the elapsed time of the voyage in days) to the date, and make any needed adjustments to the day of the month (D), the month of the year (M), and the year (Y). While doing these things, this monster also adds ET to GT (the game time, used in figuring a score). JD is used to help calculate the month and year, and can be thought of as the "Julian date" — the number of days elapsed within a particular year.

The end of line 170 will set the flag TR to zero at the end of five out of one hundred voyages. The effect of this is that Li Yuen will then consider that another payment is due — and we won't be under his "protection" until he's paid again.

RETURNing to line 1290, we next display information showing where we've arrived and how long it took us to get there. The Delay subroutine is called in order to give us time to read the message, and we GOTO line 120, where our activities in the new port o' call can start.

Before going on, let's look at the lines we've covered together in this chapter:

155 REM UPDATE SUBROUTINE (160-172)

```
160 ET = INT (ET + (ET * RND (1) / 3)):  
GT = GT + ET:  
D = D + INT (D * (ET / 360)):  
JD = JD + ET:  
IF JD > 360 THEN JD = JD - 360:  
Y = Y + 1
```

```
161 IF JD < 1 THEN JD = 1
```

```
170 M = INT ((JD - 30) / 30):  
DA = INT (((JD / 30) - INT (JD / 30)) * 30):  
IF RND (1) > .95 THEN TR = 0
```

171 IF M < 0 THEN M = 0

172 RETURN

1160 VTAB 19:

PRINT " WE'RE FIRING ON 'EM!"
GOSUB 5540:
IF INT (RND (1) * (G + 1)) + 1 > INT (RND (1)
* 4)
THEN VTAB 19:
PRINT B\$;:
HTAB 1:
PRINT "HIT 'EM!"
GOSUB 5680:
GOTO 1170

1161 VTAB 19:

PRINT " MISSED 'EM!"
GOSUB 780:
GOTO 1190

1170 NK = RND (1):

IF NK > .4 THEN IF P > = G THEN
X = INT (INT ((RND (1) * (G - 1)) + 1) / 3) +
1

1171 IF NK > .4 AND P < G THEN X = INT (RND (1) *
P) + 1

1172 IF NK < = .4 THEN VTAB 19:

PRINT " THEY'RE STILL AFLOAT."
GOSUB 780:
GOTO 1190

1180 P = P - X:
VTAB 19:
PRINT " SANK ";X;" OF 'EM, TAIPAN!":
GOSUB 5830:
IF P > 0 THEN GOSUB 5290

1190 GOSUB 1100:
IF P = 0 THEN IF RND (1) > .5 THEN 1270

1191 IF P = 0 THEN VTAB 19:
PRINT " THAT'S ALL FOR THE BUGGERS!":
GOSUB 780:
IF RND (1) > .8 THEN 1030

1192 IF P = 0 THEN 1290

1193 GOSUB 780:
GOTO 1050

1200 IF RND (1) < .8 THEN VTAB 19:
PRINT " THEY REFUSE TO PARLEY, TAIPAN.":
GOSUB 780:
GOTO 1050

1201 VTAB 19:
PRINT " THEY AGREE TO DISCUSS TERMS.":
GOSUB 780

1210 IF P > G / 2 OR RND (1) > .7 THEN VTAB 19:
PRINT " THEY OFFER TO LET US GO IN EXCHANGE
FOR";:
PRINT "ALL OUR CASH. DO WE ACCEPT, TAIPAN (Y/
N)":
GOTO 1220

1211 GOTO 1240

1220 GOSUB 60:

VTAB 19:

PRINT A\$1:

PRINT A\$1:

VTAB 18:

IF X\$ = "Y" THEN IF RND (1) > .2 AND

C > 100 + INT (RND (1) * 2000) THEN P = 0:

C = 0:

VTAB 19:

PRINT " THEY TOOK IT & RAN!":

GOSUB 780:

GOSUB 1100:

GOTO 1280

1221 IF X\$ <> "Y" THEN 1050

1222 C = 0:

VTAB 19:

PRINT " THEY STILL INTEND TO FIGHT!":

PRINT:

PRINT A\$1:

GOSUB 780

1230 GOSUB 780:

GOTO 1050

1240 VTAB 1:

HTAB 13:

```
PRINT $FL$;;
VTAB 19:
PRINT " THEY OFFER TO SURRENDER.      ":
GOSUB 780:
VTAB 19:
PRINT " DO WE LET THEM GIVE UP (Y/N)?      ":
GOSUB 60:
IF X$ = "Y" THEN 1260

1250 IF RND (1) > .5 AND P > 0
THEN E = INT ( RND (1) * P ) + 1:
P = P - E:
GOSUB 6100:
VTAB 19:
PRINT B$;;
HTAB 1:
PRINT E;" OF 'EM RAN AWAY!":
GOSUB 780:
GOSUB 1100:
GOSUB 5290:
GOTO 1050

1260 IF RND (1) > .7 THEN VTAB 19:
PRINT " THEY'RE PREPARING TO ATTACK!      ":
GOSUB 780:
GOTO 1050

1265 REM      CAPTURE (1270-1281)
```

1270 VTAB 19:

```
PRINT " WE'VE CAPTURED A BIGGER SHIP!":  
GOSUB 760:  
VTAB 19:  
PRINT " WE'RE TRANSFERRING TO IT NOW.":  
GOSUB 780:  
G = G + INT ( RND (1) * (G + 1)) + 1:  
E = SH + INT ( RND (1) * (SH + 150)) + 1:  
SH = SH + E:  
MW = MW + E:  
P = 0
```

1280 IF SR < .1 THEN 1300

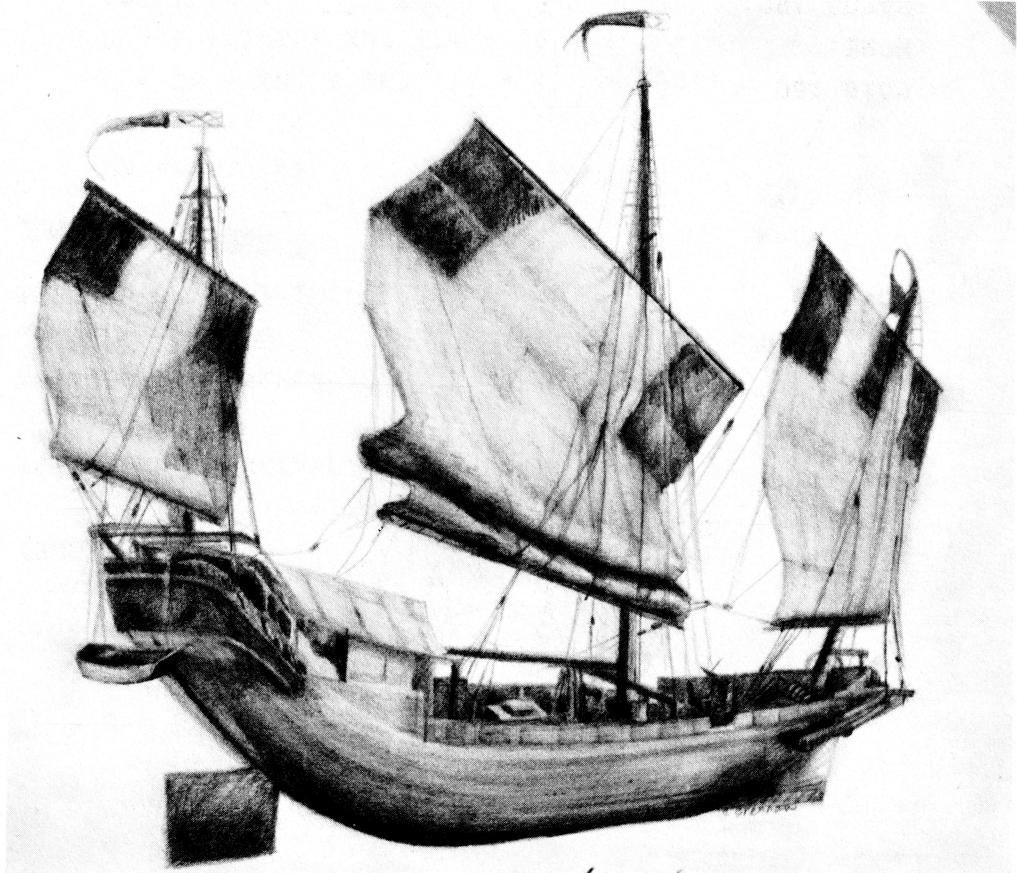
1281 IF P > 0 THEN 1050

1285 REM ARRIVAL (1290)

```
1290 CR = 0:  
SR = 1:  
L = P0:  
V(L) = V(L) + 1:  
B = 0:  
K = 0:  
GOSUB 160:  
HOME:  
PRINT:  
INVERSE:  
PRINT A$:::  
NORMAL:
```

```
PRINT " ARRIVING ";L$(L);" AFTER":  
PRINT " A VOYAGE OF ";ET;" DAYS.":  
INVERSE:  
PRINT A$:  
NORMAL:  
GOSUB 780:  
HOME:  
GOTO 120
```





Lorchha

T A I P A N

A Historical Adventure for the Apple® Computer

Ships of the China Coast

CHAPTER SEVENTEEN

The last thing the Arrival routine did was to take the program all the way back to the Main Display routine, which begins at line 120.

Line 120, we'll recall, was simple:

```
120  GOSUB 130:  
      GOTO 220
```

With "GOSUB 130", the Main Display information is displayed. Then we GOTO 220, where the Market Prices are displayed.

But the first thing that line 220 does is to GOSUB 790. Remember when, many chapters ago, we said we'd get to the Events subroutine later? Well, it's later now.

The Events subroutine is where the occasional happenings of the game are handled.

The first lines of the Events subroutine allow the player to "trade up" to a larger ship.

But what type of ships did the taipans — and the East Asian pirates — use? What actual craft might correspond to the 50-unit ship with which a player starts a game?

One book which the Authors came across in our research, is a large, beautifully illustrated work, purporting to be about "ships". Looking in the index, we found but one reference to "junks". We looked at the proper page, and found a definition something like this: "Junks are Chinese boats." That book had very little else to say about junks, though every conceivable type of Western craft was treated in detail. This was worse than a disappointingly brief treatment of junks — it was inaccurate: junks are Chinese *ships*, not boats. Any good sailor knows a boat is a craft which is small enough to be placed aboard a ship. *Sampans* are Chinese boats, while junks are seagoing or riverine ships. Perhaps the Authors should not have been surprised at the apparent ignorance of those who'd written that book about ships. After all, hardly anything about the largest nation on this planet is taken seriously by most Westerners. Junks are only one example.

The sampan (from the Chinese *san pan*, literally "three boards") is known to have existed for over three thousand years. The Shang dynasty (1766–1122 BC) had an ideograph for "boat", which was a simplified picture of a sampan. These craft still can be found in Hongkong's Aberdeen harbor, propelled efficiently by stern sweeps. A more modern form of sampan is Hongkong's "walla-walla boat", which uses a hull similar to a Western ship's gig. Also known as "Hongkong sampans", these little craft have been used continuously since the earliest days of the Crown Colony.

(One of the Authors was a young U.S. Navy sailor in the mid-1960s. One night while on a rare overnight liberty in the Crown Colony, he had — with the aid of several iron-gutted British swabs from the Singapore-based submarine, *Anchorite* — nearly drunk Kowloon dry. Returning to Hongkong island so late that the ferries had stopped operating, he can even now remember stepping into a motorized walla-walla boat. But he has no memory of crossing the harbor — or anything else, for that matter — until he awoke the next morning in his bed at the Ascot Hotel.)

Junks themselves are of uncertain age, but have been around as ocean-going vessels at least from the time of the T'ang dynasty (618–916 AD). While the Western world had fallen into feudalism and ignorance, China's T'ang empire was an enlightened civilization, with junks trading by sea throughout East Asia.

The Yuan (Mongol) dynasty (1260-1367) had huge junks longer than one hundred feet. These, along with Korean ships, were used by Kublai Khan's forces in their invasion of Japan. (The word *Kamikaze*, Japanese for "Divine Wind", originated when a terrible typhoon — said to have been called up by the Japanese Buddhist priest Nichiren — sank most of these ships, saving Japan from the Mongols.)

By the early 1400s, Chinese junk fleets were trading and exploring as far away from their homeland as India, Arabia and Africa.

Many maritime writers dismiss junks as "boats", and treat them contemptuously. (Even the English word for them — junk — suggests this contempt.) But G. R. G. Worchester, a British authority on the craft, wrote in his fascinating book, *Sail and Sweep in China*, that Chinese junks were probably at least equal to, if not better than, anything on the Earth's seas up until about four hundred years ago.

Junks were built with watertight compartments and with rudders which could be lifted in shallow water. Many could carry two or three hundred tons of cargo, yet were built so they could navigate in only three feet of water.

But the Chinese were very conservative in their naval architecture (as with everything else). Once "perfected" the design of junks remained completely unchanged for centuries. Junks are still economically important along the China coast and on China's vast inland waterway system.

When the Portuguese established themselves in Macau in the mid-1500s, they were confronted with local pirates. Since Chinese admirals of that time would rarely risk themselves against pirates, the Chinese relied upon the Portuguese to combat the sea-rovers. The Portuguese and the Chinese shipwrights of Macau, probably over a period of many years, developed a hybrid type of ship, half Western and half Chinese: the lorcha.

Lorchas were rigged with Chinese-style sails and Western-style hulls. These beautiful and hard-working little ships were faster and could haul more cargo than a junk of similar size, carrying anywhere from 40 to 300 tons of cargo.

Around the beginning of the 1800s, the Chinese arranged with the Portuguese to create a joint anti-pirate fleet, consisting of sixty Imperial Chinese war junks and six Portuguese lorchas. This fleet was to patrol the mouth of the Pearl river and to suppress piracy. Soon most of Imperial

junks had actually joined the pirates. But those six tough little lorchas defeated the pirates on their own!

The taipans of Hongkong used lorchas for carrying opium and other goods in their trading and smuggling operations along the China coast. The craft were ideal ships for rendezvousing with huge warehouse-like receiving ships hidden in isolated inlets along the coast. From these barges, fast oar-driven "dragon boats" could race the opium to shore with little fear of interception by Chinese authorities. After the Yangtze river was opened to foreigners in 1861, the versatile lorchas carried cargo as far as 600 miles up that river.

The lorcha is the only type of ship to have a war named after one of its kind. In 1856, the lorcha *Arrow* (which the British claimed was of Hongkong registry), was seized by the Chinese. When the British commenced military retaliations, the result was called "The Arrow War".

The last of the lorchas disappeared, Worcester suggests, during the Second World War. They were filled with rock and were sunk across the Yangtze river at Matung, to prevent the Japanese from using the river.

Since the lorchas were so effective against the pirates of the China Seas, it stands to reason that the pirates themselves used some other type of craft. The authors have come across references to pirates masquerading as fishermen, so it seems certain that almost all of the pirate craft were junks. The crews of pirate junks would pile fishing nets, covered with thick leather, along the sides of the craft as an effective bullet-stopping barrier. The pirate junks were mainly the types of craft which were used along Fukien and Kwantung provinces. These ships were of deeper draft, and more seaworthy, than the junks of China's northern climes. (The northern junks were built flat-bottomed in order to navigate shallow waters common to northern ports and rivers.)

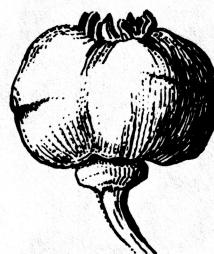
Although the graphics show no fine detail, you may like to know that the pirate ships depicted by the routines in the "Sea Action" section of this book represent large sea-going junks of Kwantung province.

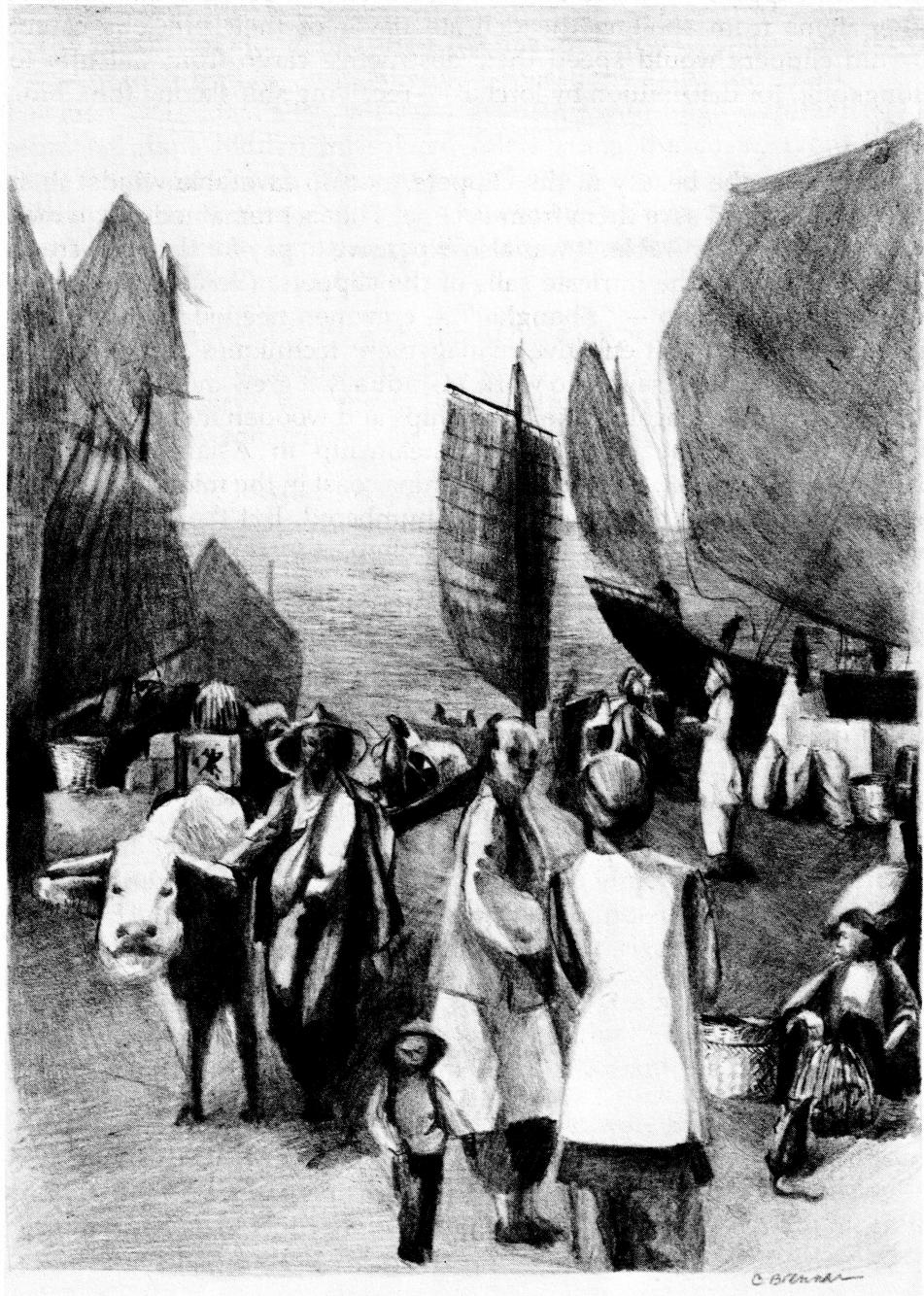
The larger ships used by the China Transfer were European and American sailing and steam craft. These ranged from little two-masted fore-and-aft rigged schooners, through grimly and ugly — but reliable — seagoing steamships, to the culmination of the Age of Sail, the magnificent China Clippers.

The clippers could, if the winds were fair, outrace any steamship of the time. Tea clippers would carry nothing *but* tea, to prevent the odors of other items from spoiling the delicate flavor of their precious cargo. Opium clippers would speed their destructive cargo from Calcutta to Hongkong, for distribution by lorchas to receiving ships along the China coast.

But neither the beauty of the clippers, nor (in favorable winds) their high speed, could save them from decline. The fact remained that winds were not always favorable. It was also expensive to pay for the large crews needed to handle the intricate sails of the clippers. (Some sea captains would simply kidnap — “Shanghai” — crewmen needed for the China trade, using such cost-effective management techniques as flogging to force the involuntary sailors to work.) Gradually it grew more difficult for clippers to compete against the “iron ships and wooden men” of the Age of Steam. Since the day the first steamship in Asian waters, the paddlewheeler, *Forbes*, arrived on the China coast in the middle of the last century, the days of the clippers were numbered. But the doomed clippers continued to ply the seas, and new clippers were being built as late as the first decades of this century.

Now the clippers are gone, the lorchas are scuttled, and the pirates were few and far between. But the junks of the China Coast endure. They were there first. Somehow, it would not be surprising if the junks outlasted the supertankers and containerships of the present day.





Waiting for the ship to come in

T A I P A N

A Historical Adventure for the Apple® Computer

Current Events

CHAPTER EIGHTEEN

Here are the first lines of the Events subroutine:

```
785 REM EVENTS SUBROUTINE (790-851)
790 IF K = 1 THEN RETURN
791 K = 1:
X = 50 + INT ( RND (1) * 100) + 1:
GN = INT ( RND (1) * 3) +1:
XP = (X + (GN * 50)) * 100:
IF C < XP OR RND (1) < .75 THEN 805
792 GOSUB 1340:
VTAB 12:
PRINT " A BROKER OFFERS TO TAKE YOUR":
PRINT "VESSEL IN TRADE FOR ONE WITH":
PRINT GN + G;" GUNS & ";X + MW;" CAPACITY"
```

```

800 PRINT "FOR ";XP;" IN CASH. WILL YOU?":
PRINT " ACCEPT (<Y> OR <N>)?";:
GOSUB 60:
VTAB 12:
IF X$ = "Y" THEN C = C - XP:
SH = SH + X:
MW = MW + X:
G = G + GN:
GOSUB 130:

```

This routine will allow the player to trade in one ship for a slightly larger one with more guns. These lines operate only one time out of four, if we just arrived in port, and if we have more than enough cash.

A "1" for the value of the flag K indicates that we've already gone through the Events subroutine this time in port, and line 790 then RETURNS us to the beginning of the Market Price routine at line 220. But if we haven't gone through Events, line 791 sets K to 1.

Line 791 next sets both how much extra capacity (X) is in the new ship, and how many more guns (GN) it has. Then the expense of the transaction (XP) is calculated. This is done by multiplying the number of additional guns by 50, adding the result to the extra hold capacity, then multiplying that sum by 100. If we then are rich enough and lucky enough, we're offered the deal.

Line 800 continues the offer display, and (if we choose to trade in our ship) the line subtracts the cost of the deal (XP) from our cash (C). It then adds the amount of the extra capacity (X) to both our available hold space (SH) and to our ship's total capacity (MW). The new guns (GN) are added to our old (G), and we GOSUB 130 to update the screen in light of the transaction.

Whether or not we're offered a larger ship, we'll continue the Events subroutine:

```
805 GOSUB 1340
```

```

810 IF C > D AND D > 2000 AND RND (1) > .7
THEN GOSUB 1340:
VTAB 12:
PRINT "YOU'VE BEEN ATTACKED AND ROBBED
BY IRON LOTUS RUFFIANS, TAIPAN!";;
C = INT (C /3):
GOSUB 760:
GOSUB 130

```

Here we see a very good argument in favor of paying off our debts! There's a chance that Elder Brother Wu's braves may rob us of one third of our cash. This can happen if: 1) our cash is greater than what we owe Wu, 2) we owe Wu more than 2000 in cash, and 3) we're unlucky. Wu's Iron Lotus society has members in all of our ports o'call. We don't have to be in Hongkong to be robbed. (Notice that our debt is not reduced if we're mugged!)

The Events subroutine now continues to line 820:

```

820 GOSUB 180:
GOSUB 860:
GOSUB 190:
GOSUB 1340:
DN = INT ((C / 2) * RND (1)):
IF RND (1) > .8 AND TR = 0
AND L <> 0 THEN VTAB 12:
PRINT "A MESSENGER FROM ";LY$;" ASKS":
PRINT "THAT YOU RETURN TO HONGKONG"

821 IF RND (0) > .8 AND TR = 0 AND L <> 0 THEN
PRINT "WITHOUT DELAY, TAIPAN.":
GOSUB 780

```

First off, line 820 has us GOSUB 180. That routine, the Price Variation subroutine, was explained in detail in Chapter 5. It operates once upon entering each port in order to randomly set the actual prices for each item of trade — GP(I) — within a range determined by the base prices for the items in that port — AP(L,I). This must be done simply because there is always some fluctuation in markets. It's also more interesting to provide some uncertainty to the player. Yet the variations in prices provided for in this subroutine are fairly minor compared to the variations from one port to the next. This fact makes it generally possible to plan a port-to-port strategy for trading.

But when line 820 next GOSUBs 860, a wider fluctuation of market prices is brought into play:

```
855 REM BEARS 'N' BULLS SUBROUTINE (860-910)
```

```
860 I = INT ( RND (1) * 6 )
```

```
870 IF RND (1) < .85 THEN RETURN
```

```
880 GP(I) = INT (GP(*)  
* ( RND (1) * 4 ) + .5)
```

```
890 VTAB 12:  
PRINT L$(L);" MARKET FORCES HAVE":  
PRINT "DRIVEN ";G$(I);" PRICES TO ";:  
PRINT GP(I);:  
PRINT "!";
```

```
900 GOSUB 780
```

```
910 RETURN
```

Out of every hundred port visits we make, about fifteen times we'll encounter larger than normal price variations in one item. That item is selected randomly in line 860.

Line 870 has us RETURN without unusual fluctuations in about 85% of our visits.

The new price of the item involved is set to anywhere from one half its "normal" price to three times normal, in line 880. (It's always possible that the price set will *still* not be very unusual.)

An announcement of the fluctuation is given by line 890. Line 900 gives a Delay subroutine call, and 910 RETURNS program flow to the Events subroutine at line 820.

There is yet a third subroutine which line 820 now calls, the Hi/Lo subroutine:

```
185 REM HI-LO SUBROUTINE (190-210)

190 FOR I = 0 TO 5:
    IF GP(I) > H(L,I) THEN H(L,I) = GP(I)

200 IF GP(I) < L(L,I) OR L(L,I) = 0 THEN L(L,I) =
    GP(I)

210 NEXT I:
    RETURN
```

The Hi/Lo subroutine keeps information up to date for the Records subroutine. The subroutine goes through a loop, comparing the current local market prices for each item — GP(I) — to the previous high and low prices for the same items. If a new record high or low price for an item in the port we're visiting is encountered, that value is placed into two-

dimensional array H(L,I) or L(L,I). After these checks are made, the program RETURNS.

Meanwhile, back at line 820, variable DN is set to a value equal to up to half the player's cash. (This variable may be used soon to determine how much Li Yuen wants for a "donation".) Next, a warning message may be given to the player, if the player's tribute to piratical Li Yuen is not "paid up". The flag TR can either equal zero, in which case we're in arrears with our "insurance policy", or TR can equal one, to indicate we're "in good hands" with Li Yuen. If Li has either never been paid tribute, or has decided that our last payment has "expired", a "please visit" message may be delivered to us upon arriving in any port except Hongkong. In these circumstances, we'll encounter one of Li's messengers about one time out of five. (Generally speaking, this is a message which should be heeded!)

Every time we arrive in Hongkong without Li Yuen's protection, we're given (assuming we have at least one hundred in cash) an opportunity to make a "contribution" to Li:

```

830 IF C > 100 AND TR = 0
      AND L = 0 THEN GOSUB 1340:
      VTAB 12:
      PRINT Y$";", LIEUTENANTS":
      PRINT "OF THE MARINER, ";LY$";", ASK IF":
      PRINT "YOU WILL DONATE ";:
      Q = DN:
      GOSUB 1330:
      PRINT "TO THE TEMPLE OF TIN HAU, THE"
831 IF C > 100 AND TR = 0 AND L = 0
      THEN PRINT "SEA GODDESS. (Y) OR (N)"
```

Using the value we set in line 820 for DN, the above lines give us a very pleasantly worded shake-down for money. (We can pay and feel warm inside — after all, we're just making a church contribution!)

The next two lines handle our input, adjustments to our cash and tribute-status, and the reaction of Mr. Yamato and Mr. Smythe:

```
840 IF C > 0 AND TR = 0 AND L = 0 THEN GOSUB 60:
IF X$ = "Y" THEN C = C - DN:
```

```
TR = 1:
```

```
GOSUB 130:
```

```
GOSUB 1340:
```

```
VTAB 12:
```

```
PRINT " ":
```

```
PRINT " ";YS$;" THANK":
```

```
PRINT " YOU, AND DEPART.":
```

```
PRINT:
```

```
PRINT A$;
```

```
841 IF C > 0 AND TR = 0 AND L = 0
```

```
THEN IF X$ <> "Y" THEN GOSUB 1340:
```

```
VTAB 12:
```

```
PRINT " ":
```

```
PRINT " ";YS$;" DEPART":
```

```
PRINT "ABRUPTLY IN A CHILLY SILENCE.":
```

```
PRINT A$;
```

Now we only need to do a little screen-cleaning, run a Delay subroutine call, and update the screen (in case we just parted with some money). We also need to RETURN, so the game can continue at line 220. Lines 850 and 851 do this:

```
850 IF (TR = 0 AND L = 0)
OR X$ = "Y" AND L = 0 THEN PRINT:
```

```
PRINT A$;:
```

```
GOSUB 780:
```

```
VTAB 13:
```

```
PRINT A$:
```

```
PRINT:
```

```
PRINT
```

851 RETURN

Give yourself a hearty pat on your back. You've now finished entering the main portion of Taipan! Only the Sea Action subroutines in the next chapter remain.

Here are this chapter's lines of BASIC code once again:

```

185 REM HI-LO SUBROUTINE (190-210)

190 FOR I = 0 TO 5:
    IF GP(I) > H(L,I) THEN H(L,I) = GP(I)

200 IF GP(I) < L(L,I) OR L(L,I) = 0 THEN L(L,I) =
    GP(I)

210 NEXT I:
    RETURN

785 REM EVENTS SUBROUTINE (790-851)

790 IF K = 1 THEN RETURN

791 K = 1:
    X = 50 + INT ( RND (1) * 100) + 1:
    GN = INT ( RND (1) * 3) +1:
    XP = (X + (GN * 50)) * 100:
    IF C < XP OR RND (1) < .75 THEN 805

792 GOSUB 1340:
    VTAB 12:
    PRINT " A BROKER OFFERS TO TAKE YOUR":
    PRINT "VESSEL IN TRADE FOR ONE WITH":
    PRINT GN + G;" GUNS & ";X + MW;" CAPACITY"

```

```
800 PRINT "FOR ";XP;" IN CASH. WILL YOU":  
PRINT " ACCEPT (<Y> OR <N>)?";  
GOSUB 60:  
VTAB 12:  
IF X$ = "Y" THEN C = C - XP: SAY "YOU GOT IT"  
SH = SH + X:  
MW = MW + X:  
G = G + GN:  
GOSUB 130  
  
805 GOSUB 1340  
  
810 IF C > D AND D > 2000 AND RND (1) > .7  
THEN GOSUB 1340:  
VTAB 12:  
PRINT "YOU'VE BEEN ATTACKED AND ROBBED  
BY IRON LOTUS RUFFIANS, TAIPAN!":  
C = INT (C / 3):  
GOSUB 760:  
GOSUB 130  
  
820 GOSUB 180:  
GOSUB 860:  
GOSUB 190:  
GOSUB 1340:  
DN = INT ((C / 2) * RND (1)): IF DN = 0 THEN  
IF RND (1) > .8 AND TR = 0 AND L <> 0 THEN VTAB 12:  
PRINT " A MESSENGER FROM ";LY$;" ASKS":  
PRINT "THAT YOU RETURN TO HONGKONG"  
  
821 IF RND (0) > .8 AND TR = 0 AND L <> 0 THEN  
PRINT "WITHOUT DELAY, TAIPAN.":  
GOSUB 780
```

```
830 IF C > 100 AND TR = 0
    AND L = 0 THEN GOSUB 1340:
    VTAB 12:
    PRINT YS$;" LIEUTENANTS":
    PRINT "OF THE MARINER, ";LY$;" ASK IF":
    PRINT "YOU WILL DONATE ";:
    Q = DN:
    GOSUB 1330:
    PRINT "TO THE TEMPLE OF TIN HAU, THE"

831 IF C > 100 AND TR = 0 AND L = 0
    THEN PRINT "SEA GODDESS. (Y) OR (N)"

840 IF C > 0 AND TR = 0 AND L = 0 THEN GOSUB 60:
    IF X$ = "Y" THEN C = C - DN:
    TR = 1:
    GOSUB 130:
    GOSUB 1340:
    VTAB 12:
    PRINT " ":
    PRINT " " ;YS$;" THANK":
    PRINT " YOU, AND DEPART. ":
    PRINT:
    PRINT A$:

841 IF C > 0 AND TR = 0 AND L = 0
    THEN IF X$ <> "Y" THEN GOSUB 1340:
    VTAB 12:
    PRINT " ":
    PRINT " " ;YS$;" DEPART":
    PRINT "ABRUPTLY IN A CHILLY SILENCE. ":
    PRINT A$:
```

```
850 IF (TR = 0 AND L = 0)
OR X$ = "Y" AND L = 0 THEN PRINT:
PRINT A$;;
GOSUB 780:
VTAB 13:
PRINT A$:
PRINT:
PRINT
```

```
851 RETURN
```

```
855 REM BEARS 'N' BULLS SUBROUTINE (860-910)
```

```
860 I = INT ( RND (1) * 6 )
```

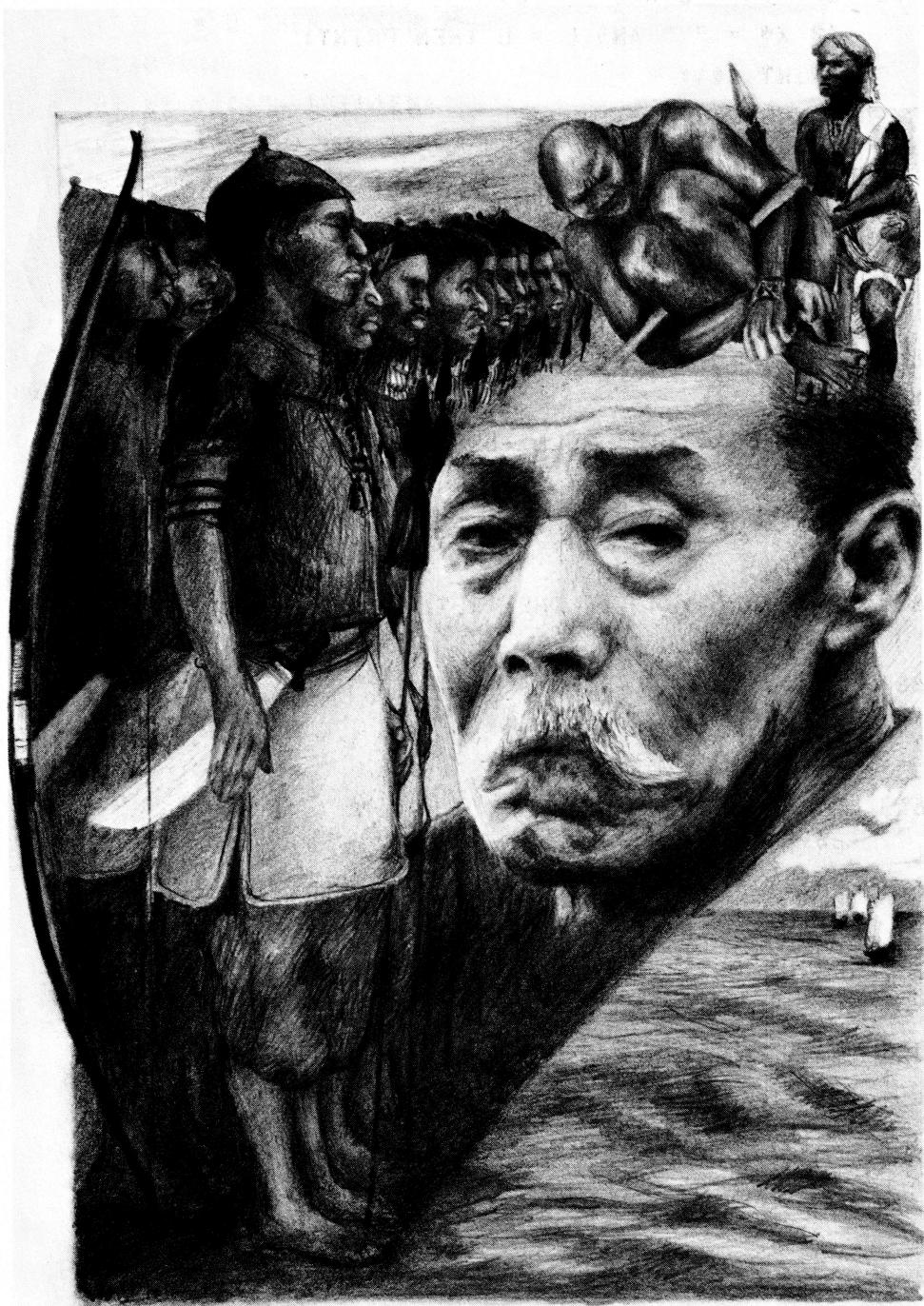
```
870 IF RND (1) < .85 THEN RETURN
```

```
880 GP(I) = INT (GP(I)
* ( RND (1) * 4 ) + .5)
```

```
890 VTAB 12:
PRINT L$(L);" MARKET FORCES HAVE":
PRINT "DRIVEN ";G$(I);" PRICES TO ";
PRINT GP(I);:
PRINT "!";
```

```
900 GOSUB 780
```

```
910 RETURN
```



Nicholas Iquan

C. Brennan

T A I P A N

A Historical Adventure for the Apple® Computer

Action at Sea

CHAPTER NINETEEN

These subroutines are a simple method of giving some visual excitement to our pirate counters.

Using the Apple II in BASIC, it is very difficult to operate high-resolution graphics, especially with any kind of animation speed. So what we have here is an example of compromises: we've used your Apple's normal characters, like "-", "/", "\", ":" and "0", to build and move a fair representation of a pirate junk.

These subroutines, which are called by GOSUBs mainly in the Pirates routine, will display the arrival and the attack of a pirate ship. If you're lucky, you'll also get to see pirate ships run away from you, or even sink. Or you might witness yourself sinking!

By now, you must be either very anxious to RUN Taipan, or very weary of typing this game into your machine. (Maybe both!) The Authors will spare you convoluted explanations in this chapter, and will only tell briefly what the routines do.

Here's the first routine, which is called by line 50 in the Initialized routine when the program first starts. This sets up the shape of the pirate ship:

```
4995 REM SEA ACTION SUBROUTINES (5000-6360)

4997 REM GRAPHICS INITIALIZATION (5000-5120)

5000 DIM CH$(14),CN$(14)

5005 FOR V1 = 0 TO 12

5010 READ CH$(V1)

5020 CN$(V1) = CH$(V1)

5030 NEXT V1

5040 DATA V1    :      "
5050 DATA      "-----"
5060 DATA      "-----\:\  "
5070 DATA      " :/   ----- \\"   "
5075 DATA      " //  ----- \\\\"   "
5080 DATA      "///  ----- \\\\"   "
5085 DATA      "///  ----- \\"   "
5090 DATA      "///  ----- :\  "
5100 DATA      "//  ----- :  "
```

```

5105 DATA   ("/:-----:-: "
5110 DATA    " :--: : :000: "
5111 DATA    ": \-----/ / "
5112 DATA    " \-----/ / "
5120 RETURN

```

You can already see, in the DATA lines, what the pirate junk looks like, can't you?

Now, here's the first Sea Action subroutine which the Pirates routine calls when the enemy is first sighted:

```
5290 REM    PIRATES ARRIVE (5300-5370)
```

```
5300 GOSUB 5950
```

```
5305 IF ID <> 1 THEN CH$(0) = "====: RETN" "
```

```
5310 FOR A = 1 TO 10:
    VTAB 1:
    PRINT A$;;
    NEXT A
```

```
5320 FOR A = 1 TO 30
```

```
5340 FOR B = 0 TO 13
```

```
5350 VTAB B + 1:
    HTAB 40 - A:
    PRINT LEFT$(CH$(B),A),;" "
```

5360 NEXT B:

NEXT A

5370 RETURN

The Pirates Arrive subroutine makes the pirate ship sail into view from the right side of the screen. If the ship is one of Li Yuen's, a banner is shown atop the highest mast.

The following subroutine shows the blazing guns of the pirate ship:

5380 REM PIRATES FIRE (5380-5440)

5390 FOR A = 1 TO INT (RND (1) * 6) + 1

5410 A1 = INT (RND (1) * 10) + 1

5420 IF A1 = 5 THEN 5410

5430 HTAB 16 + A1:

VTAB 11:

PRINT "**"

5432 PRINT "AG"

5433 HTAB 16 + A1:

VTAB 11:

PRINT " "

5435 NEXT A

5440 RETURN

Notice the “^G” in line 5432 above. It’s there to let us hear the pirates firing off each shot. (Remember to type it using the Control and G keys simultaneously.)

The “Cannon Hit” subroutine makes our beeper sound off to show either that we’ve been hit or that the pirates have:

```
5540 REM CANNON HIT (5550-5560)

5550 PRINT "^G"

5560 RETURN
```

This subroutine punches holes in the image of the pirate ship, if we’ve hit it:

```
5680 DAMAGE 'EM (5730-5760)

5730 I = INT ( RND (1) * 17) + 2:
    I1 = INT ( RND (1) * 9) + 2

5740 CH$(I1) = LEFT$ (CH$(I1), I - 1) + " "
    + MID$ (CH$(I1), I + 1, LEN (CH$(I1)))

5750 HTAB 10:
    VTAB I1 + 1:
    PRINT CH$(I1)

5755 GOSUB 780

5760 RETURN
```

There are few greater pleasures than seeing this one work! The pirate junk slides down into the ocean:

```
5830 REM     PIRATES SINK (5840-5940)
```

```
5840 FOR A = 0 TO 12
```

```
5850 VTAB A + 1:
```

```
    PRINT A$;
```

```
5870 FOR A1 = 0 TO 12 - A
```

```
5880 VTAB 2 + A + A1:
```

```
    HTAB 10:
```

```
    PRINT CH$(A1)
```

```
5890 NEXT A1
```

```
5910 NEXT A
```

```
5920 VTAB 14:
```

```
    PRINT A$;
```

```
5930 GOSUB 5950
```

```
5940 RETURN
```

If we've sunk one pirate junk, this subroutine makes the next junk look undamaged:

```
5950 REM     REPAIR DAMAGE (5960-5990)j
```

```
5960 FOR A = 0 TO 13
```

```
5970 CH$(A) = CN$(A)
```

5980 NEXT A

5990 RETURN

Next to the "Pirates Sink" subroutine, this one's our favorite. It makes the pirates disappear off the left side of the screen:

6100 REM PIRATES DEPART (6110-6210)

6110 FOR A = 9 TO 0 STEP - 1

6130 FOR B = 0 TO 13

6140 HTAB A+ 1:

VTAB B + 1:

PRINT CH\$(B); " "

6150 NEXT B:

NEXT A

6160 FOR A = 22 TO 1 STEP - 1

6180 FOR B = 0 TO 13

6190 HTAB 1:

VTAB B + 1:

PRINT RIGHT\$(CH\$(B),A); " "

6200 NEXT B:

NEXT A

6210 RETURN

If we run from the pirates, we show the pirate junk receding to the right side of the screen. (If we don't manage to get away from any or all of the attackers, we'll soon see a junk catching up with us.) Here's the subroutine:

```

6220 REM WE PULL AWAY (6230-6280)
6230 FOR A = 30 TO 1 STEP -1
6250 FOR B = 0 TO 13
6260 HTAB 40 - A:
    VTAB B + 1:
    PRINT " "; LEFT$(CH$(B),A); CHR$(8);" ";
6270 NEXT B:
    NEXT A
6280 RETURN

```

Ouch! We see bubbles floating up the screen as we head for Davy Jones':

```

6290 REM WE'RE SUNK! (6300-6360)
6300 FOR A = 1 TO 17
6310 FOR B = 1 TO 100:
    NEXT B
6320 VTAB 24:
    HTAB INT (RND (1) * 40) + 1:
    PRINT "0"

```

6350 NEXT A

6360 RETURN

There you have it. Congratulations on typing in a Contextual Computer Game!



T A I P A N

A Historical Adventure for the Apple® Computer

Bibliography

APPENDIX A

The Authors would like to express their appreciation to the Public Library system. It is a treasure of our nation. May it remain free.

Here are some of the books which either inspired us, or otherwise aided us in our project:

Tai-Pan, by James Clavell, Dell

Dynasty, by Robert S. Elegant, Fawcett

Sail and Sweep in China, by G.R.G. Worchester, Her Majesty's Stationery Office

Rascals in Paradise, by James Michener and A. Grove Day, Random House

Sailing through China, by Paul Theroux, Houghton Mifflin Company

Historical and Geographical Dictionary of Japan, by E. Papinot, Charles E. Tuttle Company

China: Adventures in Eyewitness History, by Rhoda Hoff, Henry Z. Walck, Inc.

The Art of War, by Sun Tzu (edited by James Clavell), Delacorte Press

T A I P A N

A Historical Adventure for the Apple® Computer

Playing Taipan

APPENDIX B

Taipan is meant to be a challenging game. A player may sometime find that some things which happen in the game are "unfair". Even making the right decisions doesn't always work. For instance, paying Li Yuen his tribute does not guarantee that he won't attack us without warning. Likewise, the cruel sea may cause our ship to founder, ending our budding mercantile career early.

But in the real world, people *are* sometimes as unpredictable as Li Yuen. And the sea has never cared for human concepts of justice. We must do the best that we can, within the limits of success which our intelligence allows. If an "unfair" event kills us off in Taipan, remember that it is *just a game*. We pick up the pieces and start over again.

Taipan needs very little in the way of playing instructions. Everything which happens is self-explanatory. The main guides to use when playing the game are common sense and the lessons of experience. This is how the real taipans operated in a world alien to them.

People are often tempted to "cheat", even in single-player games. It's possible, of course, to cheat in *Taipan*. But the issue of *who* is cheated is important. A player can get twenty-humpzillion guns and three-quadshillion in cash, simply by input variable values while in the "command" mode. But why? What kind of *game* would we then have?

Taipan is a game which puts the player in the shoes of a China Trader of the 1800's. Those were not clean shoes — they were stained with blood and opium. The Authors do not intend that players emulate the old taispans in real life but rather to obtain some measure of understanding of a real historical context through the medium of the microcomputer. This is an opportunity to "re-live" the past which was impossible for earlier generations — and the Contextual Computer Games of the future will doubtless make our efforts appear puny.

Index

Arrival routine, 131, 171
Arrow War, 184
ASCII, 65

Bears 'n' Bulls subroutine, 190
Big Number Subroutine, 39
Bombay, 15
Boston Tea Party, 13
British, 15
Buy/Sell routine, 48

Calcutta, 15
Cape of Good Hope, 25
Captain China, 140
Captain del Cano, 22
Capture routine, 165, 170
Cebu, 22
Cheating, 47, 64
China, 1
China clippers, 185
China Trade, 1, 2
Chinese Taels, 29
Ch'ing Dynasty, 13
Clean Up subroutine, 78
Contextual computer game, 5
Commissioner Lin Tse-hsu, 16
Crown Colony, 20
Cursor, 64

Delay subroutine, 61
Dollars Mex, 22
Dragon boats, 184
Dutch, 23
 posts, 23
 East India Company, 23

Elder Brother Wu, 11, 29, 41
Elliot, Captain Charles, 20
Embark routine, 129
Events subroutine, 187

Flag, 47
Formosa, 140

Game, 29
Game time, 29
Get String, 45
Godown, 17
Godown subroutine, 85
 Item choice, 86
Grand Canal, 21

Han Empire 13
Harris, Townsend, 21
Hi/Lo subroutine, 191
Hongkong, 20
 Victoria, 20
Hsu, Paul, 13
Huai, 21

Initialization, 9
Input Error, 80
Input Error subroutine, 46
Intramuros, 22
Iron Lotus Triad, 111
Item Choice subroutine, 49
Iquan, Nicholas, 140, 141

Japan, 1, 14
Jingals, 144
Junks, 183

Kamikaze, 183
Koxinga, 140

Lapu-Lapu, 22
Lender menu, 113
Lender subroutine, 112
Liverpool, 25
Li Yuen, 11

- Loops, 28
- nested, 27
- Lorcha, 183
- Macau, 20, 140, 183
- Magellan, Ferdinand, 22
- Main Display subroutine, 37
- Malay, 17
- Manchu Empire, 141
- Manila, 22
- Market Menu Input routine, 44
- Ming Dynasty, 13, 141
- Min Kingdom, 21
- Muslim, 22
- Nicholas Iquan, 140, 141
- No-Can-Do subroutine, 61
- Opium War, 16
- Other Options routine, 60, 87
- Pearl River, 20
- Pepper, 14
- Perry, Commodore, 1, 21
- Philippines, 21
- Pirates, 147
- Pirates routine, 147
- Polo, Marco, 14
- Port Choice subroutine, 97
- Portuguese, 21
- Price Variation subroutine, 28
- Queen Victoria, 16
- Raffles Hotel, 23
- Raffles, Sir Thomas Stamford, 23
- Random number generator, 30
- Randomizer subroutine, 30, 34
- Records subroutine, 96
- menu, 97
- Restaurant French, 19
- Roaring Forties, 25
- Saigon, 24
- Sampans, 182
- Schall, Father Adam, 13
- Scientific notation, 39
- Ship Status subroutine, 148
- Silk, 15
- Singapore, 19
- Smythe, 151
- Sooper Dooper Number Scooper subroutine, 62
- Spain, 21
- Spices, 1
- Taipan, 1
- Taipan geography, 19
- Tea, 13
- That's All Folks routine, 123
- Transaction Quantity routine, 67
- Treaty of Kanagawa, 21
- Treaty ports, 20
- Triads, 2
- Update subroutine, 172
- Victoria, 20
- Voyage routine, 131
- storm, 133
- Walla-walla boat, 183
- Whole numbers, 67
- Yamato, 151
- Yangtse river, 21
- Yellow river, 21



T A I P A N

*A Historical Adventure for the Apple® Computer
is available on disk!*

For just \$4.95 plus shipping and handling, we'll send you a disk that contains the Taipan program. The disk runs on the Apple II Plus, IIe, or IIc.

Use the handy order form below to order your Taipan adventure.

Please send me the disk containing the Taipan adventure (#7764-5). I am enclosing a check or money order for \$4.95 per disk plus \$2.50 for shipping and handling for each order.

Quantity	Price	Total
_____	\$4.95	_____
NJ and CA residents add tax shipping and handling		_____
total amount		\$2.50

Name (print) _____

Address _____

City _____ State _____ Zip _____

Prices subject to change. Offer good in USA only. Allow 4-6 weeks for delivery.

Send your check or money order to:

Hayden Book Company
Department TP
10 Mulholland Drive
Hasbrouck Heights, NJ 07604